Gaussian Process Deep Belief Networks: A Smooth Generative Model of Shape with Uncertainty Propagation

Di Martino, Alessandro

Award date:
2019

Awarding institution:
University of Bath

Gaussian Process Deep Belief Networks: A Smooth Generative Model of Shape with Uncertainty Propagation

Alessandro Di Martino
Department of Computer Science
University of Bath

This thesis is submitted for the degree of
Doctor of Philosophy

2019

Abstract

The shape of an object is an important characteristic for many vision problems such as segmentation, detection and tracking. Being independent of appearance, it is possible to generalise to a large range of objects from only small amounts of data. However, shapes represented as silhouette images are challenging to model due to complicated likelihood functions leading to intractable posteriors. In this work we present a generative model of shapes which provides a low-dimensional latent encoding which importantly resides on a smooth manifold with respect to the silhouette images. The proposed model propagates uncertainty in a principled manner allowing it to learn from small amounts of data and providing predictions with associated uncertainty. Our experiments show that the proposed model provides favourable quantitative results compared with the state-of-the-art while simultaneously providing a representation that resides on a low-dimensional interpretable manifold.

Acknowledgements

First, I would like to thank my supervisor, Dr Neill D.F. Campbell, for his insight and constant guidance through all the stages of this work. I am also grateful to have been co-supervised by Dr Carl Henrik Ek who also supported and guided me. This work would not have been possible without their many inspiring ideas, their deep knowledge in the field and their enthusiasm. Also, a special thanks to my colleague and collaborator Erik Bodin for the many insightful discussions and ideas that have spawned, which were very valuable for my work.

I would like to thank Prof Mike Tipping and Dr Tom Fincham Haines for their feedback at the initial stages of my work.

I am grateful to the examining committee: Dr Karteek Alahari and Dr Tom Fincham Haines, for accepting to review this thesis and for my viva examination.

On a more personal note I would like to thank all my colleagues and friends, too many to be mentioned here singularly, who in so many ways have made my studies and my time in Bath a wonderful experience.

Finally, I would like to gratefully acknowledge the University of Bath for awarding me a studentship to fund my PhD studies, and the *Centre for the Analysis of Motion, Entertainment Research and Applications (CAMERA)* that provided travel funding for conferences.

Dedication

I dedicate this thesis to my special patroness saint, Mary, mother of Jesus. And through her, I give thanks and glory to God, the One who gave me everything: life, intelligence, freedom, love and forgiveness.

For my thoughts are not your thoughts,
nor are your ways my ways, says the Lord.
For as the heavens are higher than the earth,
so are my ways higher than your ways
and my thoughts than your thoughts.

Isaiah 55:8–9

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data Manifolds

High-dimensional data poses serious challenges to machine learning approaches. To alleviate this problem and make computations tractable it is reasonable to assume that the data lies on or close to a lower dimensional manifold embedded in the actual higher-dimensional space. To exemplify this, we can imagine the space of all possible images of a fixed size; of all possible combinations of pixel values, only a tiny fraction actually represents natural images, this fraction is confined nearby regions of high probability density which have much lower dimensionality.

To further exemplify this concept, it is easier to think in low dimensions and imagine a two-dimensional manifold as a paper sheet on which some datapoints have been fixed at random locations. If we bend the paper sheet we can easily see the points arranged in the three-dimensional space. The locations of the points can be encoded using a three-dimensional coordinate system, however because the intrinsic structure of the data is two-dimensional, in order to recover the locations of the points in the high-dimensional space, if we knew the mapping between the manifold and the data space, it would just be enough to use the two-dimensional coordinate system of the manifold and the mapping, effectively compressing the dimensionality of the data.

In reality, high-dimensional datapoints do not normally lie exactly on low-dimensional manifolds, but in order to reduce the dimensionality of the data it is reasonable to assume that they reside nearby these

low-dimensional high probability density regions (Bengio, Courville and Vincent, 2013).

Smooth and Interpretable Manifolds

The focus of this thesis is to present a method able to learn a *smooth* mapping from a low-dimensional manifold space to a high-dimensional data space. We refer to this as the *generative* mapping. Specifically, we tackle the problem of learning a smooth manifold of binary silhouettes.

The *generative* aspect of the model implies that there is a well-defined low-dimensional space that can be sampled; *smoothness* of this mapping from the latent space to the data space is another key property that we desire for our generative model: if two points in the latent space are "close" to each other, a smooth function maps them to "close" points in the data space as well.

The space of silhouette images is challenging to work with as it is not smooth in terms of a representation as pixels. A transformation that we would consider semantically smooth might correspond to a drastic change in pixel values. For instance, we can imagine binary images of horses: two semantically close silhouettes might be very far apart in pixel space simply because of the fact that the leg positions of the two silhouettes do not match perfectly. Our goal is to learn a smooth low-dimensional representation of silhouette images such that silhouettes can be generated in a natural manner; intuitively, this implies that the output varies smoothly as we generate from close locations in the latent space.

The specific form of the low-dimensional manifold with its guaranteed smoothness makes the latent space easy to work with and more readily interpretable. We note that, *interpretability* is a broad concept in machine learning (Lipton (2018)), in this work we use this term meaning this desirable property of the latent space that makes it easier to understand where to sample from. This kind of interpretability that we are interested in is also achieved by exploiting the posterior predictive uncertainty information provided by Gaussian processes which gives an indication of where to sample in the latent space to obtain low-variance samples (we elaborate more on the predictive uncertainty providing a mathematical formulation in Chapter 2 where we introduce Gaussian processes). We acknowledge

that there are limitations to this definition of *interpretability*. Although smoothness of the latent space and posterior predictive uncertainty make it more readily interpretable, a low-dimensional representation of complex data can still contain ambiguity because of loss of information.

Uncertainty Propagation

As data is at a premium, we want to learn a fully probabilistic model that allows us to propagate uncertainty throughout the generative process. This allows us to learn from *smaller amounts of data* and also associate a quantified uncertainty to its predictions. This uncertainty allows the model to be used as a building block in larger models.

The results of our proposed model challenge the current trend in unsupervised learning towards maximum likelihood training of increasingly large parametric models with increasingly large datasets. We demonstrate that by propagating uncertainty throughout the model our approach outperforms two standard generative deep learning models, a Variational Autoencoder (VAE) (Kingma and Welling, 2013) and a Generative Adversarial Network (InfoGAN) (Chen et al., 2016) with comparable architectures and can achieve good performance with far smaller training datasets.

Combining Gaussian Processes and Deep Belief Networks

In this thesis we revisit a few classic machine learning models with complementary properties. On the one hand, parametric models such as restricted Boltzmann machines (RBMs) (Smolensky, 1986) are particularly interesting as they are stochastic as well as generative and can be stacked easily into *deeper* models such as deep belief networks (DBNs); these can be trained in a greedy fashion, layer by layer (Hinton and Salakhutdinov, 2006), and can approximate a probability distribution on visible units. DBNs (*i.e.*, stacked RBMs), in addition, learn deep representations by composing features learned by the lower layers, yielding progressively more abstract and flexible representations at higher layers and often leading to more expressive and efficient models compared to shallow alternatives (Bengio and LeCun, 2007).

However, a DBN suffers from a number of limitations. Firstly, it does not guarantee a smooth representation in the learned latent space. Secondly, the DBN does not provide an explicit representation of the uncertainty (this can only arise as a byproduct of the propagation of multiple samples). Thirdly, a DBN does not provide any explicit generative process from a manifold, in fact the standard way to sample from a DBN is to start from a training example and perform iterations of Gibbs sampling.

On the other hand, the Gaussian Process Latent Variable Model (GPLVM) (Lawrence, 2005), which is a non-parametric model, combines a Gaussian process (GP) prior with a likelihood function in order to learn a low-dimensional representation. By specifying a prior that encourages smooth functions, a smooth latent representation can be recovered. However, to make inference tractable the likelihood is also chosen to be Gaussian, which does not reflect the statistics of natural images. Further, even though the mapping from the latent space is non-linear, the posterior is linear in the observed space. This makes the GPLVM unsuitable for modelling images. To circumvent this problem one can compose hierarchies of Gaussian processes (Damianou and Lawrence, 2013), however, these models are inherently difficult to train.

The characteristics of the DBN and GPLVM can be considered complementary, where the DBN excels the GPLVM fails and vice versa; the aim of this thesis is to present a model that combines the best properties of both.

Unfortunately, combining the two models into a single one by simply stacking a GPLVM on top of a DBN would not preserve uncertainty propagation throughout the joint model. The ability to learn from a small dataset expands the applicability of a model to domains where there is a lack of available data or where collection of data is costly or time-consuming, therefore it is essential to make efficient use of the data by correctly propagating uncertainty throughout the model. Furthermore, a simple stack of the two models would pose a challenge to training: while the GPLVM is a non-parametric model trained by optimising an objective function, the DBN is a parametric model, with non-differentiable binary units, and is trained with contrastive divergence (Sec. 2.1.2).

Goals and Contributions

The novelty of our work can be summarised in the following contributions:

1. A model (which we call GPDBN) that combines the properties of a smooth, interpretable manifold for synthesis with a data specific likelihood function (a deep structure) capable of decomposing images into an efficient representation while propagating uncertainty throughout the model in a principled manner.

2. We propose a single objective function to train the model end to end using backpropagation with the same complexity as a standard feed-forward neural network.

3. We also show that the model is able to learn from very little data, outperforming current generative deep learning models, as well as scaling linearly to larger datasets by the use of mini-batching.

4. A key feature of the GPDBN model is uncertainty propagation. The use of dropout units (Srivastava et al., 2014) is a popular way to propagate uncertainty in neural network models. In this work we have also experimented with some alternative dropout model variants and provided quantitative comparisons with the proposed GPDBN.

5. Assessing unsupervised (shape) models is inherently a difficult task as there is no well-defined measure to quantify how well a model can generalise from the training data to novel shapes. Chapter 7 is devoted to discussing this more in depth.

Applications

The GPDBN provides a compact way for capturing and describing the variation of a class of objects, moreover, shapes can be synthesised efficiently and in a natural manner from a smooth manifold that is interpretable. Although in this work we do not focus on concrete applications of shape model priors, some examples of the practical problems that would possibly benefit from the properties of the GPDBN model include any applications where it is important to have a statistical shape model (especially if the

data is scarce): object recognition (Toshev, Makadia and Daniilidis, 2009), object detection (Ferrari, Jurie and Schmid, 2010; Toshev, Taskar and Daniilidis, 2012), pose estimation (Elgammal and Lee, 2004; Yang and Ramanan, 2011; Reinbacher, Ruther and Bischof, 2010), feature localisation (Li, Gu and Kanade, 2009; Gu and Kanade, 2008), image segmentation (Patenaude et al., 2011; Grosgeorge et al., 2013), object reconstruction (Yemez and Schmitt, 2004). Also, any applications where it is important to capture and represent smooth changes in shape representations such as: tracking (Li et al., 2016; Cremers, 2006), shape synthesis (Kalogerakis et al., 2012), guided sketching and modelling tools (Andre and Saito, 2011).

Outline of the Thesis

As mentioned earlier, the aim of this thesis is to introduce the GPDBN: a generative model that provides a smooth manifold of shapes with uncertainty propagation. In Chapter 4 we fulfil this aim by providing an ample description and the mathematical details of the model.

We have devoted Chapter 2 to provide the necessary background, specifically, we describe restricted Boltzmann machines and Gaussian processes in detail as they are important building blocks for our model.

In Chapter 3 we describe related methods and models, which we then compare against our model in Chapter 5; here we provide various qualitative and quantitative experimental results.

Chapter 6 is focused on uncertainty propagation in the context of dropout architectures, here we compare our baseline model with some dropout variants, providing also experimental results.

In Chapter 7 we discuss the idea of model's generalisation from training data to test data, providing our views on good assessment and similarity measures.

Finally, Chapter 8 contains the discussion and conclusions including avenues for future work.

Associated Publication and Software

The work presented in this thesis is based on our published paper (Di Martino et al., 2018) co-authored by A. Di Martino, E. Bodin, C. H. Ek and N. D. F. Campbell.

The accompanying software developed to implement the models described in Chapter 4 is publicly available on GitHub (Di Martino et al., 2019). The library also includes code to implement Gaussian Process Latent Variable Models (Lawrence, 2005) and deep belief networks.

An interactive demo comparing our GPDBN and the GPLVM can be downloaded from the ancillary resources of the Arxiv paper repository[1].

[1]GPDBN paper repository on Arxiv: https://arxiv.org/abs/1812.05477

Chapter 2

Background

Understanding deep belief networks and Gaussian processes is essential to better understand the GPDBN model presented in this thesis (Chapter 4), we have dedicated this chapter to provide a formal introduction of the key aspects of these models.

Specifically, in Section 2.1, we introduce restricted Boltzmann machines; these constitute the building blocks of deep belief networks (Sec. 2.1.3), *i.e.*, the layers of a deep belief network (DBN) are restricted Boltzmann machines and each layer in the stack can be trained separately using contrastive divergence (Sec. 2.1.2). DBNs are stochastic neural networks that can model (binary) image data using efficient hierarchical ("deep") representations. Importantly, in order to sample from a deep belief network, one must provide an input example to condition on; in other words, these models do not define a proper generative process that would allow us to easily generate from a low-dimensional manifold. This issue also related to the conditional independence assumption of the latent dimensions of the restricted Boltzmann machine, as we show more formally in Section 2.1.

Later in this chapter we also introduce Gaussian processes (Sec. 2.2) and the related unsupervised Gaussian Process Latent Variable Model (GPLVM)(Sec. 2.2.4). These methods are not suitable to model images because of the assumption of a Gaussian likelihood. On the other hand, Gaussian processes ensure a smooth generative mapping from the latent space to the data space thanks to the choice of smooth covariance functions (Sec. 2.2.1), as well as providing posterior uncertainty information for

predictions, thus, providing an interpretable and easy to work with generative latent space. All these aspects are presented in detail in Section 2.2.

The good properties of Gaussian processes and deep belief networks can be combined into a single model, we show how this can be achieved in Chapter 4, where we introduce the GPDBN model building on the background knowledge of the present chapter.

2.1 Restricted Boltzmann Machines

The restricted Boltzmann machine (RBM), or Harmonium (Smolensky, 1986) is a generative stochastic neural network that learns a probability distribution over a vector of random variables. The graphical model of the RBM is an undirected bipartite graph, consisting of a set of visible random variables (or units): $\boldsymbol{v}$, and a set of hidden units $\boldsymbol{h}$ (Fig. 2.1). Typically, all variables are binary (Bernoulli), taking on values from $\{0, 1\}$.

The RBM model specifies a probability distribution over both the visible and hidden variables jointly as

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} \exp\left(-E(\boldsymbol{v}, \boldsymbol{h})\right) \tag{2.1.1}$$

which defines a Gibbs distribution with energy function

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{v}^\top \boldsymbol{W} \boldsymbol{h} - \boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h} \ , \tag{2.1.2}$$

where $\boldsymbol{W}$, $\boldsymbol{b}$, $\boldsymbol{c}$ are the parameters of the model: $\boldsymbol{W}$ is a linear weight matrix and $(\boldsymbol{b}, \boldsymbol{c})$ are bias vectors for the visible and hidden units respectively. The normalising constant Z is the, computationally intractable, sum over all possible random vectors $\boldsymbol{v}$ and $\boldsymbol{h}$ (it ensures that $\sum_{\boldsymbol{v},\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) = 1$).

The bipartite structure of the model (*i.e.*, the graph has no visible-visible or hidden-hidden connections, as shown in Fig. 2.1), affords efficient Gibbs sampling from the visible units given the hidden variables (or vice versa). The conditional distribution of the hidden units given the visible ones, and vice versa, factorise as each set of variables are conditionally

independent given the other:

$$p(\boldsymbol{h} \mid \boldsymbol{v}) = \prod_{j=1}^{H} p(h_j \mid \boldsymbol{v}), \tag{2.1.3}$$

$$p(\boldsymbol{v} \mid \boldsymbol{h}) = \prod_{i=1}^{V} p(v_i \mid \boldsymbol{h}) \ . \tag{2.1.4}$$

We show below the derivation of the conditional distribution $p(\boldsymbol{h} \mid \boldsymbol{v})$:

$$\begin{aligned}
p(\boldsymbol{h} \mid \boldsymbol{v}) &= \frac{p(\boldsymbol{v}, \boldsymbol{h})}{p(\boldsymbol{v})} = \frac{\frac{1}{Z}\exp(-E(\boldsymbol{v}, \boldsymbol{h}))}{\frac{1}{Z}\sum_{\boldsymbol{h}'}\exp(-E(\boldsymbol{v}, \boldsymbol{h}'))} = \frac{\exp(-E(\boldsymbol{v}, \boldsymbol{h}))}{\sum_{\boldsymbol{h}'}\exp(-E(\boldsymbol{v}, \boldsymbol{h}'))} \\
&= \frac{\exp(-E_1(\boldsymbol{v}, h_1))\exp(-E_2(\boldsymbol{v}, h_2))\ldots\exp(-E_H(\boldsymbol{v}, h_H))}{\sum_{h'_1}\exp(-E_1(\boldsymbol{v}, h'_1))\sum_{h'_2}\exp(-E_2(\boldsymbol{v}, h'_2))\cdots\sum_{h'_H}\exp(-E_1(\boldsymbol{v}, h'_H))} \\
&= \prod_{j=1}^{H}\frac{\exp(-E_j(\boldsymbol{v}, h_j))}{\sum_{h'_j}\exp(-E_j(\boldsymbol{v}, h'_j))} = \prod_{j=1}^{H}\frac{\exp(\boldsymbol{v}^\top \boldsymbol{w}_j h_j + \boldsymbol{v}^\top \boldsymbol{W} + c_j h_j)}{\sum_{h'_j}\exp(\boldsymbol{v}^\top \boldsymbol{w}_j h'_j + \boldsymbol{v}^\top \boldsymbol{W} + c_j h'_j)} \\
&= \prod_{j=1}^{H}\frac{\exp(\boldsymbol{v}^\top \boldsymbol{w}_j h_j + c_j h_j)}{\sum_{h'_j}\exp(\boldsymbol{v}^\top \boldsymbol{w}_j h'_j + c_j h'_j)} = \prod_{j=1}^{H} p(h_j \mid \boldsymbol{v}) \ .
\end{aligned} \tag{2.1.5}$$

In Eq. 2.1.5 $\boldsymbol{w}_j$ is the j^{th} column of $\boldsymbol{W}$, and H is the number of hidden units. For binary units we have that $h_j \in \{0, 1\}$, so the probability $p(h_j = 1 \mid \boldsymbol{v})$ is given by the sigmoid function[1]:

$$\begin{aligned}
p(h_j = 1 \mid \boldsymbol{v}) &= \frac{\exp(\boldsymbol{v}^\top \boldsymbol{w}_j + c_j)}{\sum_{h'_j}\exp(\boldsymbol{v}^\top \boldsymbol{w}_j h'_j + c_j h'_j)} = \frac{\exp(\boldsymbol{v}^\top \boldsymbol{w}_j + c_j)}{\exp(0) + \exp(\boldsymbol{v}^\top \boldsymbol{w}_j + c_j)} \\
&= \text{Sigmoid}(\boldsymbol{v}^\top \boldsymbol{w}_j + c_j) \ .
\end{aligned} \tag{2.1.6}$$

Similarly:

$$p(v_i = 1 \mid \boldsymbol{h}) = \text{Sigmoid}(\boldsymbol{w}_i \boldsymbol{h} + b_i) \ . \tag{2.1.7}$$

Other types of units can be used instead of binary units. For instance, to have Gaussian hidden units it is enough to use the following energy

[1] $\text{Sigmoid}(x) = \exp(x)/(\exp(x) + 1)$

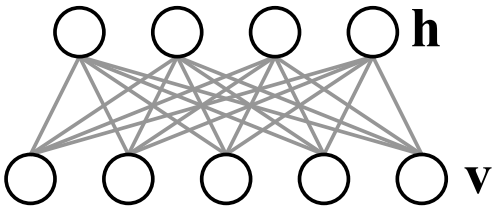


Figure 2.1: Graphical model of an RBM.

function (Hinton, 2012):

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{V} b_i v_i + \sum_{j=1}^{H} \frac{(h_j - c_j)^2}{2\sigma_j^2} - \sum_{i=1}^{V}\sum_{j=1}^{H} \frac{1}{\sigma_j} v_i h_j w_{ij} \ . \tag{2.1.8}$$

This leads to the following Gaussian conditional distribution over h_j:

$$p(h_j = 1 \mid \boldsymbol{v}) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\Big\{ -\frac{1}{2\sigma_j^2}[h_j - (c_j + \sigma_j \sum_{i=1}^{V} v_i w_{ij})]\Big\} \ . \tag{2.1.9}$$

2.1.1 Learning

The parameters of the model can be learned by minimising the maximum likelihood function in an approximate way (as we describe later in Section 2.1.2).

To derive the negative log likelihood function it useful to introduce the *free energy* function:

$$F(\boldsymbol{v}) = -\log \sum_{\boldsymbol{h}} \exp\left(-E(\boldsymbol{v}, \boldsymbol{h})\right) \ . \tag{2.1.10}$$

We note that $F(\boldsymbol{v})$ can be calculated efficiently, indeed the sum over $\boldsymbol{h}$ in Eq. 2.1.10 is a compact notation for

$$F(\boldsymbol{v}) = -\boldsymbol{b}^\top \boldsymbol{v} - \sum_{j=1}^{H} \log \sum_{h_j} \exp\left(\boldsymbol{v}^\top \boldsymbol{w}_j h_j + c_j h_j\right) \ .^{2} \tag{2.1.11}$$

Using the free energy we can write the marginal distribution $p(\boldsymbol{v})$ in the

[2]For Gaussian hidden units the sum over h_i is replaced by an integral.

following way:

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) = \frac{\exp\left(-F(\boldsymbol{v})\right)}{Z} \ , \tag{2.1.12}$$

here $Z = \sum_{\boldsymbol{v}} \exp(-F(\boldsymbol{v}))$. Also, we note that $\boldsymbol{h}$ does not appear explicitly.

Given a training dataset $\mathcal{D} = \{\boldsymbol{v}_n\}_{n=1}^N$ and using $\boldsymbol{\theta}$ as a placeholder for a set of parameters of the model, the negative log-likelihood function of $\boldsymbol{\theta}$ given $\boldsymbol{v}_i$ is

$$\begin{aligned} -L(\boldsymbol{\theta} \mid \boldsymbol{v}_i) &= -\log p(\boldsymbol{v}_i \mid \boldsymbol{\theta}) = -\log \frac{\exp(-F(\boldsymbol{v}_i \mid \boldsymbol{\theta}))}{Z(\boldsymbol{\theta})} \\ &= -\log \exp(-F(\boldsymbol{v}_i \mid \boldsymbol{\theta})) + \log Z(\boldsymbol{\theta}) \\ &= F(\boldsymbol{v}_i \mid \boldsymbol{\theta}) + \log Z(\boldsymbol{\theta}) \ . \end{aligned} \tag{2.1.13}$$

The gradient of the negative log-likelihood with respect to $\boldsymbol{\theta}$ can be calculated easily:

$$\begin{aligned} -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta} \mid \boldsymbol{v}_i) &= \nabla_{\boldsymbol{\theta}} F(\boldsymbol{v}_i \mid \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \log Z(\boldsymbol{\theta}) \\ &= \nabla_{\boldsymbol{\theta}} F(\boldsymbol{v}_i \mid \boldsymbol{\theta}) + \frac{\nabla_{\boldsymbol{\theta}} Z(\boldsymbol{\theta})}{Z(\boldsymbol{\theta})} \\ &= \nabla_{\boldsymbol{\theta}} F(\boldsymbol{v}_i \mid \boldsymbol{\theta}) - \frac{1}{Z(\boldsymbol{\theta})} \sum_{\boldsymbol{v}} \exp[-F(\boldsymbol{v} \mid \boldsymbol{\theta})] \nabla_{\boldsymbol{\theta}} F(\boldsymbol{v} \mid \boldsymbol{\theta}) \\ &= \nabla_{\boldsymbol{\theta}} F(\boldsymbol{v}_i \mid \boldsymbol{\theta}) - \sum_{\boldsymbol{v}} p(\boldsymbol{v} \mid \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} F(\boldsymbol{v} \mid \boldsymbol{\theta}) \ . \end{aligned} \tag{2.1.14}$$

If we take the expected value of both sides of Eq. 2.1.14 under the distribution of the data $p(\boldsymbol{v}_i)$ we obtain:

$$\mathbb{E}_{p(\boldsymbol{v}_i)}[-\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta} \mid \boldsymbol{v}_i)] = \mathbb{E}_{p(\boldsymbol{v}_i)}[\nabla_{\boldsymbol{\theta}} F(\boldsymbol{v}_i \mid \boldsymbol{\theta})] - \mathbb{E}_{p(\boldsymbol{v}|\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} F(\boldsymbol{v} \mid \boldsymbol{\theta})] \ . \tag{2.1.15}$$

In the right-hand side of Eq. 2.1.15, the first expectation is referred to as the *data* term, and the second is the *model* term (because it is an expectation under $p(\boldsymbol{v} \mid \boldsymbol{\theta})$). Unfortunately, the model term is intractable because it involves a sum over all possible values of $\boldsymbol{v}$.

Learning is difficult since direct calculation of the gradients of the log likelihood with respect to the parameters requires the intractable

computation in Eq. 2.1.15. Currently, in practice, the approximate maximum-likelihood contrastive divergence algorithm is used (Carreira-Perpiñán and Hinton, 2005).

2.1.2 Contrastive Divergence

Contrastive divergence (CD) is an approximate maximum-likelihood learning algorithm that has been shown to work well with RBMs (Carreira-Perpiñán and Hinton, 2005). It is based on the following ideas:

1. Replacing the expectation of the model term under the distribution of all possible $\boldsymbol{v}$ in Eq. 2.1.15 with a point estimate (*i.e.*, a single sample): $\boldsymbol{v}_k$.

2. Obtaining $\boldsymbol{v}_k$ by performing k iterations of Gibbs sampling starting at some training datapoint $\boldsymbol{v}_0$:

$$\begin{aligned} \boldsymbol{v}_0 &\sim p(\boldsymbol{v}_i) \ , \\ \boldsymbol{h}_0 &\sim p(\boldsymbol{h} \mid \boldsymbol{v}_0) \ , \\ \boldsymbol{v}_1 &\sim p(\boldsymbol{v} \mid \boldsymbol{h}_0) \ , \\ \boldsymbol{h}_1 &\sim p(\boldsymbol{h} \mid \boldsymbol{v}_1) \ , \\ &\cdots \\ \boldsymbol{v}_k &\sim p(\boldsymbol{v} \mid \boldsymbol{h}_{k-1}) \ . \end{aligned} \tag{2.1.16}$$

Contrastive divergence is a biased algorithm because of the above-mentioned approximations, however, in practice the bias is typically small, and CD learning tends to converge close to a maximum-likelihood optimum Carreira-Perpiñán and Hinton (2005).

One variation of contrastive divergence is *persistent contrastive divergence* (PCD) (Tieleman, 2008). In PCD the sampling chain to obtain $\boldsymbol{v}_k$ is never reinitialised at every k iterations at a training datapoint $\boldsymbol{v}_0$, in fact, it is initialised only once, when training starts.

With the standard CD algorithm, in order to obtain a sample from the model, the sampling chain is initialised at a training point $\boldsymbol{v}_0$ and then a sample is generated after k iterations. If k is large the sampling chain runs for many iterations, taking a long time, on the other hand, if

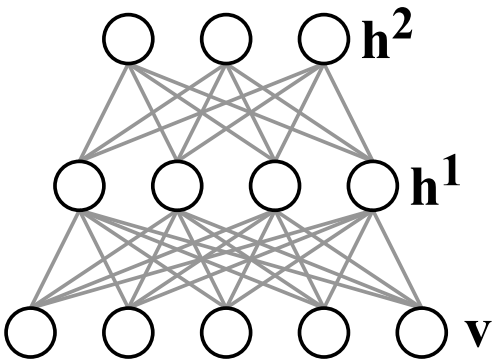


Figure 2.2: Graphical model of a DBN.

k is too small, $\boldsymbol{v}_k$ will not move too far from the starting point $\boldsymbol{v}_0$. The intuition behind persistent CD is that if the parameter updates are small enough (*e.g.*, if learning rate is small) the model distribution will not change much after each parameter update, so, the sampling chain can be safely reinitialised at the state in which it ended after the last parameter update (Tieleman, 2008).

2.1.3 Deep Belief Networks

When multiple layers of RBMs are stacked on top of each other they form a deep belief network (Fig. 2.2). Hinton and Salakhutdinov (2006) demonstrated that a DBN can be trained in a greedy fashion, layer by layer. Essentially, the samples (activations) from the hidden units of a trained layer are used as the data to train the next layer in the stack.

By stacking multiple layers, high layers compose low-level features from lower layers to form more abstract and complex features. Representations become increasingly richer and more abstract as we move up in the layer hierarchy and higher-order dependencies are captured by the hidden units. This often leads to more expressive and efficient models compared to shallow ones (Bengio and LeCun, 2007).

Sampling

Sampling from an RBM proceeds by conditioning on some input data and performing a Gibbs sample for the hidden units. Subsequently, a Gibbs sample can be drawn for the visible units by conditioning the hidden

units on this sample. This process is then repeated for a number of cycles. Since a DBN is a stack of RBMs, this process has to be repeated for all layers; the output of one layer becomes the input to condition on for the next layer. In this way, an input datapoint can be propagated up and down the network.

Limitations

Although a DBN is good at learning low-dimensional stochastic representations of high-dimensional data, it has three key drawbacks:

1. It lacks a directed generative sampling process from a well-defined latent representation. In order to generate a sample one must condition on some input data and propagate it through the network back and forth until a sample from the lowest layer is obtained. For instance, if we wanted to generate directly "from the top" from a DBN with just 10 uppermost units (that is, by simply conditioning on the 10 upper units and without propagating up through the network any input datapoint), it would not be clear how to choose a good 10-dimensional configuration in this latent space to generate the corresponding high-dimensional sample from the visible units. (Actually, in DBNs, the number of hidden units is typically much larger than 10.)

2. There is no explicit representation of the uncertainty associated to latent points. However, uncertainty can be implicitly obtained through the propagation of multiple samples (point estimates) at each layer.

3. A side effect of the conditional independence assumption of Equations 2.1.3 and 2.1.4 is that the correlations between the hidden units of the top layer of a DBN are not captured; this is because each latent dimension is independent. Most importantly, a DBN does not, therefore, give any guarantee about learning a smooth latent space.

2.2 Gaussian Processes

A Gaussian process (GP) can be thought of as an *infinite*-dimensional Gaussian distribution. While the Gaussian distribution is specified by a mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, defining a probability distribution over a finite-dimensional random vector $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, the Gaussian process is specified by a mean function $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$, defined over an infinite index set, and specifies a distribution over functions:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')), \text{ where} \tag{2.2.1}$$

$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \tag{2.2.2}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))] \ . \tag{2.2.3}$$

Loosely speaking, a function is just an infinitely long vector of evaluations over the whole index set. Although working with infinite objects may seem impractical, a key feature of the Gaussian process is the *marginalisation property*. That is, for any finite set of input points $\boldsymbol{X} = \{\boldsymbol{x}_n\}_{n=1}^N$, the marginal distribution over the vector of the corresponding function evaluations $\boldsymbol{f} = \{f(\boldsymbol{x}_n)\}_{n=1}^N$ is a joint Gaussian:

$$p(\boldsymbol{f}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{m}, \boldsymbol{K}_{NN}) \ . \tag{2.2.4}$$

In Eq. 2.2.4, $\boldsymbol{m}$ is a vector of shape $N \times 1$, obtained by evaluating the mean function at each input point. Similarly, the covariance matrix $\boldsymbol{K}_{NN}$ (of shape $N \times N$) is constructed by evaluating the covariance function at every pair of input points.

2.2.1 Covariance Functions

The covariance function (or simply *kernel*) is a function k that maps a pair of input points $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ into $\mathbb{R}$ ($\mathcal{X} \in \mathbb{R}^D$ is the set of possible input points). The kernel plays a key role in the Gaussian process model, because it encodes prior knowledge about the latent function. A kernel function normally depends on a few parameters which are learned at training time. Valid kernel functions must give rise to valid covariance

matrices, that is, symmetric and positive semidefinite matrices (more formally, a symmetric matrix $\boldsymbol{K}$ is said positive semidefinite if and only if $\boldsymbol{x}^\intercal \boldsymbol{K} \boldsymbol{x} \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^D \setminus \mathbf{0}$). For instance, a commonly used kernel is the *squared exponential* (SE), also known as *Radial Basis Function* (RBF):

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{2\ell^2}\right) , \tag{2.2.5}$$

here σ and ℓ are the two kernel parameters; the former determines the scaling of the output, the latter controls the length-scale of the input.

2.2.2 Inference

Given a training dataset $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, a *GP prior* can be used to infer the values of the function at some test points $\boldsymbol{X}_* = \{\boldsymbol{x}_{*,j}\}_{j=1}^J$[3] conditioned on the observations. We specify our *a priori* knowledge about the function by choosing a mean and covariance function. Importantly, since we usually do not have any prior knowledge about the true mean function we can take the mean function $m(\boldsymbol{x})$ to be zero (in practice, this is not a bad assumption as we can normalise the observations to have zero mean). Thanks to the marginalisation property we can then easily construct a joint Gaussian distribution over the random vector of function evaluations $\boldsymbol{f}$ (corresponding to the observed points $\boldsymbol{X}$) and the random vector $\boldsymbol{f}_*$ (corresponding to the test points $\boldsymbol{X}_*$), $p(\boldsymbol{f}_*, \boldsymbol{f} \mid \boldsymbol{X}_*, \boldsymbol{X})$, which can also be written (more explicitly) in the following way:

$$p\left(\begin{bmatrix}\boldsymbol{f} \\ \boldsymbol{f}_*\end{bmatrix}\right) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}\boldsymbol{K}_{NN} & \boldsymbol{K}_{N*} \\ \boldsymbol{K}_{*N} & \boldsymbol{K}_{**}\end{bmatrix}\right) , \tag{2.2.6}$$

here, $\boldsymbol{K}_{N*}$ is a shorthand for $k(\boldsymbol{X}, \boldsymbol{X}_*)$, that is, a $\mathbb{R}^{N \times J}$ covariance matrix constructed by evaluating the covariance function at all the pairs of observed and test points. $\boldsymbol{K}_{NN}$ and $\boldsymbol{K}_{**}$ are constructed in a similar way, and $\boldsymbol{K}_{*N}$ is just the transpose of $\boldsymbol{K}_{N*}$.

In order to do inference about $\boldsymbol{f}_*$ we need to condition on $\boldsymbol{f}$. The conditional of a Gaussian distribution is Gaussian and its mean and

[3] The subscript asterisk denotes test points, *i.e.*, points whose corresponding function evaluations are not observed and do not belong to the training set.

covariance are well-known and can be found easily in reference tables (Rasmussen, Carl Edward and Williams, Christopher K.I. (2006, Eq. A6)):

$$p(\boldsymbol{f}_* \mid \boldsymbol{f}, \boldsymbol{X}_*, \boldsymbol{X}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \text{ where} \tag{2.2.7}$$

$$\boldsymbol{\mu} = \boldsymbol{K}_{*N}\boldsymbol{K}_{NN}^{-1}\boldsymbol{f} \tag{2.2.8}$$

$$\boldsymbol{\Sigma} = \boldsymbol{K}_{**} - \boldsymbol{K}_{*N}\boldsymbol{K}_{NN}^{-1}\boldsymbol{K}_{N*} \ . \tag{2.2.9}$$

Hence the posterior GP is:

$$f(\boldsymbol{x}) \mid \mathcal{D} \sim \mathcal{GP}(m_{\mathcal{D}}(\boldsymbol{x}), k_{\mathcal{D}}(\boldsymbol{x}, \boldsymbol{x}')), \text{ where} \tag{2.2.10}$$

$$m_{\mathcal{D}}(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{X})k(\boldsymbol{X}, \boldsymbol{X})^{-1}\boldsymbol{f} \tag{2.2.11}$$

$$k_{\mathcal{D}}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \boldsymbol{X})k(\boldsymbol{X}, \boldsymbol{X})^{-1}k(\boldsymbol{X}, \boldsymbol{x}) \ . \tag{2.2.12}$$

A common modelling choice is to assume that the observed function values are a noisy version of the true function values. In practice, we assume that every observation y_n in the training set includes some additive independent and identically distributed Gaussian noise $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$:

$$y_n = f_n + \epsilon_n \ . \tag{2.2.13}$$

This implies a factorised Gaussian distribution over the random vector $\boldsymbol{y}$:

$$p(\boldsymbol{y} \mid \boldsymbol{f}) = \prod_{n=1}^{N} p(y_n \mid f_n) = \mathcal{N}(\boldsymbol{f}, \sigma^2\boldsymbol{I}) \ , \tag{2.2.14}$$

where $\boldsymbol{I}$ is the identity matrix (in this thesis we also refer to Eq. 2.2.14 as the *Gaussian likelihood*). The joint model of the noisy observations $\boldsymbol{y}$ and the test function evaluations $\boldsymbol{f}_*$ is then

$$p(\boldsymbol{f}_*, \boldsymbol{y} \mid \boldsymbol{X}_*, \boldsymbol{X}) = \int p(\boldsymbol{y} \mid \boldsymbol{f}) p(\boldsymbol{f}_*, \boldsymbol{f} \mid \boldsymbol{X}_*, \boldsymbol{X}) \, d\boldsymbol{f} \ , \tag{2.2.15}$$

which can also be written as follows:

$$p\left(\begin{bmatrix}\boldsymbol{y} \\ \boldsymbol{f}_*\end{bmatrix}\right) = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix}\boldsymbol{K}_{NN} + \sigma^2\boldsymbol{I} & \boldsymbol{K}_{N*} \\ \boldsymbol{K}_{*N} & \boldsymbol{K}_{**}\end{bmatrix}\right) \ . \tag{2.2.16}$$

In Eq. 2.2.16, the mean of $\boldsymbol{y}$ is zero because both the mean of $\boldsymbol{f}$ and the mean of $\boldsymbol{\epsilon}$ are zero. The covariance of $\boldsymbol{y}$ is $\boldsymbol{K}_{NN} + \sigma^2 \boldsymbol{I}$ because $\boldsymbol{f}$ and $\boldsymbol{\epsilon}$ are independent. This can be easily shown by calculating the covariance of $\boldsymbol{y}$:

$$\begin{aligned}
\mathrm{Cov}(\boldsymbol{y}, \boldsymbol{y}) &= \mathbb{E}[(\boldsymbol{y} - \mathbb{E}[\boldsymbol{y}])(\boldsymbol{y} - \mathbb{E}[\boldsymbol{y}])^\top] - \mathbb{E}[(\boldsymbol{y} - \mathbb{E}[\boldsymbol{y}])]\mathbb{E}[(\boldsymbol{y} - \mathbb{E}[\boldsymbol{y}])^\top] \\
&= \mathbb{E}[\boldsymbol{y}\boldsymbol{y}^\top] - \mathbb{E}[\boldsymbol{y}]\mathbb{E}[\boldsymbol{y}^\top] \\
&= \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon})(\boldsymbol{f} + \boldsymbol{\epsilon})^\top] \\
&= \mathbb{E}[\boldsymbol{f}\boldsymbol{f}^\top + \boldsymbol{f}\boldsymbol{\epsilon}^\top + \boldsymbol{\epsilon}\boldsymbol{f}^\top + \boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top] \\
&= \mathrm{Var}(\boldsymbol{f}) + 2\mathrm{Cov}(\boldsymbol{f}, \boldsymbol{\epsilon}) + \mathrm{Var}(\boldsymbol{\epsilon}) \\
&= \boldsymbol{K}_{NN} + \sigma^2 \boldsymbol{I} \ .
\end{aligned} \tag{2.2.17}$$

The conditional distribution over $\boldsymbol{f}_*$ is

$$p(\boldsymbol{f}_* \mid \boldsymbol{y}, \boldsymbol{X}_*, \boldsymbol{X}) = \frac{\int p(\boldsymbol{y} \mid \boldsymbol{f}) p(\boldsymbol{f}_*, \boldsymbol{f} \mid \boldsymbol{X}_*, \boldsymbol{X}) \, d\boldsymbol{f}}{p(\boldsymbol{y} \mid \boldsymbol{X})} \ . \tag{2.2.18}$$

This conditional distribution is Gaussian and its mean and covariance can be easily found in reference tables (Rasmussen, Carl Edward and Williams, Christopher K.I. (2006, Eq. A6)):

$$p(\boldsymbol{f}_* \mid \boldsymbol{y}, \boldsymbol{X}_*, \boldsymbol{X}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \text{ where} \tag{2.2.19}$$

$$\boldsymbol{\mu} = \boldsymbol{K}_{*N}(\boldsymbol{K}_{NN} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y} \tag{2.2.20}$$

$$\boldsymbol{\Sigma} = \boldsymbol{K}_{**} - \boldsymbol{K}_{*N}(\boldsymbol{K}_{NN} + \boldsymbol{\sigma}^2 \boldsymbol{I})^{-1} \boldsymbol{K}_{N*} \ . \tag{2.2.21}$$

The above conditional distribution (Eq. 2.2.19) is commonly called *predictive distribution* because it is used to make predictions, with $\boldsymbol{\mu}$ being the vector of the predictive means (corresponding to the test points $\boldsymbol{X}_*$), and the predictive covariance matrix $\boldsymbol{\Sigma}$ capturing the related probabilistic uncertainties. This distribution can be generalised into the following posterior GP, which differs from the GP of Eq. 2.2.10 only in the additional

observation noise:

$$f(\boldsymbol{x}) \mid \mathcal{D} \sim \mathcal{GP}(m_{\mathcal{D}}(\boldsymbol{x}), k_{\mathcal{D}}(\boldsymbol{x}, \boldsymbol{x}')), \text{ where} \tag{2.2.22}$$

$$m_{\mathcal{D}}(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{y} \tag{2.2.23}$$

$$k_{\mathcal{D}}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}k(\boldsymbol{X}, \boldsymbol{x}) \ . \tag{2.2.24}$$

2.2.3 Learning

Training a Gaussian process corresponds to learning its *hyperparameters*, *i.e.*, the parameters of the mean and covariance functions. The probability distribution of the data given the set of hyperparameters $\boldsymbol{\theta}$ is

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \int p(\boldsymbol{y} \mid \boldsymbol{f}, \boldsymbol{\theta}) p(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta}) \, d\boldsymbol{f} \ . \tag{2.2.25}$$

Eq. 2.2.25 is referred to as *marginal likelihood* because $\boldsymbol{f}$ is marginalised out. The distribution $p(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta})$ is our zero-mean Gaussian prior over function values $\boldsymbol{f}$: $\mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_{NN})$, and $p(\boldsymbol{y} \mid \boldsymbol{f}, \boldsymbol{\theta})$ is Gaussian as well because of the noisy observation assumption of Eq. 2.2.13: $\mathcal{N}(\boldsymbol{f}, \sigma^2 \boldsymbol{I})$. Since the right-hand side of Eq. 2.2.25 is a product of two Gaussian distributions, the result is also a Gaussian dristribution (Rasmussen, Carl Edward and Williams, Christopher K.I. (2006, Eqs. A7, A8)). This Gaussian marginal over $\boldsymbol{y}$ can be directly derived from Eq. 2.2.16 with the marginalisation property:

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_{NN} + \sigma^2 \boldsymbol{I}) \tag{2.2.26}$$

Thus, the log marginal likelihood has the following form:

$$\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^\top (\boldsymbol{K}_{NN} + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{y} - \frac{1}{2}\log |\boldsymbol{K}_{NN} + \sigma^2 \boldsymbol{I}| - \frac{N}{2}\log 2\pi \ . \tag{2.2.27}$$

The function of Eq. 2.2.27 is non-linear in the hyperparameters and has no closed form solution, for this reason optimisation techniques must be used.

2.2.4 GPLVM

The Gaussian Process Latent Variable Model (GPLVM) (Lawrence, 2005) is a non-linear dimensionality reduction method that uses Gaussian pro-

cesses in an *unsupervised* fashion to map some *latent* variables $\boldsymbol{X} \in \mathbb{R}^{N \times Q}$, to observed data $\boldsymbol{Y} \in \mathbb{R}^{N \times D}$ (we note that, for dimensionality reduction, $Q \ll D$). Each observed datapoint $\boldsymbol{y}_n$ is assumed to be generated by a corresponding latent vector $\boldsymbol{x}_n$ as follows:

$$\boldsymbol{y}_n = \boldsymbol{f}(\boldsymbol{x}_n) + \boldsymbol{\epsilon}_n \ , \tag{2.2.28}$$

where, $\boldsymbol{\epsilon}_n$ is a vector (with shape $1 \times D$) of zero-mean independent Gaussian noise: $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$. Each component $f^d(\boldsymbol{x})$ of the D-dimensional function $\boldsymbol{f}(\boldsymbol{x})$ is an independent draw from the same zero-mean Gaussian process prior:

$$f^d(\boldsymbol{x}) \sim \mathcal{GP}(\boldsymbol{0}, k(\boldsymbol{x}, \boldsymbol{x}')) \ . \tag{2.2.29}$$

Hence, the marginal likelihood is

$$p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \prod_{d=1}^{D} p(\boldsymbol{y}^d \mid \boldsymbol{X}, \boldsymbol{\theta}) = \prod_{d=1}^{D} \mathcal{N}(\boldsymbol{y}^d \mid \boldsymbol{0}, \boldsymbol{K}_{NN} + \sigma^2 \boldsymbol{I}) \ , \tag{2.2.30}$$

where $\boldsymbol{y}^d$ is the d^{th} column of $\boldsymbol{Y}$.

Unlike the standard GP model, in the GPLVM, the objective is to infer the distribution over $\boldsymbol{X}$ (because $\boldsymbol{X}$ is latent) as well as the mapping from $\boldsymbol{X}$ to $\boldsymbol{Y}$. So, a prior distribution is specified for the latent matrix $\boldsymbol{X}$, and the joint model of $\boldsymbol{X}$ and $\boldsymbol{Y}$ becomes:

$$p(\boldsymbol{Y}, \boldsymbol{X} \mid \boldsymbol{\theta}) = p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}) p(\boldsymbol{X}) \ . \tag{2.2.31}$$

Learning

To train the model, the logarithm of Eq. 2.2.31 can be jointly optimised with respect to the hyperparamters and $\boldsymbol{X}$. This approach, proposed by Lawrence (2005), gives the maximum a posteriori (MAP) estimate of $\boldsymbol{X}$. A MAP estimate of $\boldsymbol{X}$, however, does not include any uncertainty information on $\boldsymbol{X}$, as the MAP is a point estimate and not a distribution over $\boldsymbol{X}$. As a remedy to this issue, M. K. Titsias and N. D. Lawrence (2010) use an approximate Bayesian procedure to train the model. In this

Bayesian GPLVM, $\boldsymbol{X}$ is marginalised out:

$$p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \int p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}) p(\boldsymbol{X})\, d\boldsymbol{X} \quad . \tag{2.2.32}$$

Unfortunately though, because of the non-linearity of X in the inverse of the covariance matrix $\boldsymbol{K}_{NN} + \sigma^2 \boldsymbol{I}$, it is not possible to optimise the marginal likelihood $p(\boldsymbol{Y} \mid \boldsymbol{\theta})$ directly, so a variational lower bound is minimised instead.

Limitations

The main benefit of the GPLVM is that it is very easy to ensure a smooth mapping from the latent representations to the observed data. Further, due to the principled uncertainty propagation of the GP, all predictions have an associated uncertainty. Importantly, however even though the mapping between latent points $\boldsymbol{X}$ and training data $\boldsymbol{Y}$ can be non-linear, the relationship between the predicted mean and the training data is linear (Eq. 2.2.23). Due to this linearity, the GPLVM, as well as a GP, is inherently not suitable for modelling image data. This limitation of the model is also made clear by the Gaussian likelihood assumption of Eq. 2.2.14.

2.3 Summary

In Section 2.1 and 2.2 we have introduced restricted Boltzmann machines and Gaussian processes, describing in more detail their features, that is, respectively, the capacity to learn efficient representations of binary image data, and the ability of learning a smooth low-dimensional generative manifold with principled uncertainty information. Building on this background, in Chapter 4, we introduce our GPDBN model that combines the good sides of both methods into a single model.

Chapter 3

Related Work

Modelling of shape is important for many computer vision tasks. It is beyond the scope of this thesis to make a complete review of the topic, we refer the reader to the comprehensive work of Taylor, Twining and Davies (2008). In our work we focus on recent unsupervised statistical models that operate directly on the pixel domain. Interest in these models was revived by the Shape Boltzmann Machine (SBM) of Eslami et al. (2014) and they have been shown to be useful for a variety of vision applications (Eslami and Williams, 2012; Kirillov et al., 2016; Tsogkas et al., 2015). These deep models can also be readily extended into the 3D domain, *e.g.*, by recent work on 3D ShapeNets (Wu et al., 2015).

3.1 Desirable Properties

Table 3.1 highlights the desirable properties of the most closely related previous works. We have identified four advantageous properties: (i) It is well known that pixel silhouettes are not well modelled by a Gaussian likelihood. (ii) The utility of an unsupervised shape model is well described by the properties of its latent representation. Ensuring a smooth manifold opens up a number of applications to data in the pixel domain that previously required custom representations, *e.g.*, interactive drawing (Turmukhambetov et al., 2015). (iii) A fully generative model ensures that there is a well-defined space that can be sampled. (iv) Correctly propagating uncertainty is vital to perform data efficient learning, for example when data is scarce or expensive to obtain.

	Non-Gaussian Likelihood	Explicit Smooth Low-Dim Manifold	Fully Generative	Propagates Uncertainty
GPLVM		✓	✓	✓
GPLVMDT		✓	✓	∼
DBN	✓			✓
SBM	✓			✓
VAE	✓	∼	✓	
InfoGAN	✓	∼	✓	
ShapeOdds	✓		✓	✓
GPDBN	✓	✓	✓	✓

Table 3.1: Summary of properties of related models.

3.2 Non-parametrically-guided Autoencoder

An autoencoder is a type of artificial neural network which can be seen as consisting of two different components: an *encoder* network which reduces the dimensionality of the input by converting it into a low-dimensional latent point (or *code*), and a *decoder* network which takes the code as the input and generates a high-dimensional datapoint (*i.e.*, a *reconstruction*, because it tries to reconstruct the input) (Fig. 3.1). Assuming that the data space is $\mathcal{X} = \mathbb{R}^D$, and latent space $\mathcal{Z} = \mathbb{R}^Q$, the encoder is a function $f_e(\boldsymbol{x}; \boldsymbol{\theta}) : \mathcal{X} \to \mathcal{Z}$, where $\boldsymbol{\theta}$ represents a set of parameters for the encoder. Similarly, the decoder is a function $f_d(\boldsymbol{z}; \boldsymbol{\phi}) : \mathcal{Z} \to \mathcal{X}$, and $\boldsymbol{\phi}$ is the decoder's set of parameters. Given a training dataset $\mathcal{D} = \{\boldsymbol{x}_n\}_{n=1}^N$, the parameters of the autoencoder can be learned jointly by minimising the squared Euclidean norm between the input and its reconstruction:

$$L(\boldsymbol{\theta}, \boldsymbol{\phi}) = \|\boldsymbol{x}_n - f_d(f_e(\boldsymbol{x}_n; \boldsymbol{\theta}); \boldsymbol{\phi})\|^2 \ . \tag{3.2.1}$$

Autoencoders are often used as tools to learn a useful latent representation of the data for different purposes such as classification and data exploration. However, as the dimensionality of the input data increases, irrelevant factors of variation dominate the input distribution (this is known as the *curse of dimensionality* problem), therefore this affects the quality of learned low-dimensional representations as well.

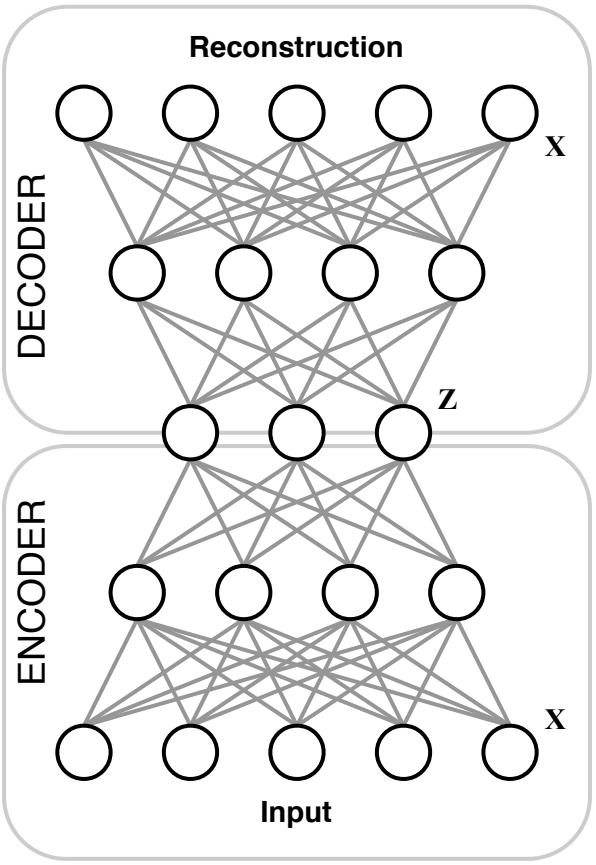


Figure 3.1: A graphical representation of the autoencoder model, where $\boldsymbol{Z}$ represents the latent space and $\boldsymbol{X}$ the observed (or data) space.

Partial Supervision

It is possible to encourage the autoencoder to learn better latent representations, for specific purposes, by using partial supervision. Bengio et al. (2007), for example, *guide* the latent representation of an autoencoder by greedily training each hidden layer with a mixed criterion that combines unsupervised and supervised learning. This is simply achieved by connecting the latent layer to a logistic regression classifier so that the layer is jointly trained to predict label information as well as to reconstruct the input.

There are, however, two main drawbacks to this approach. First, the choice of a specific parametric model (the logistic classifier) represents a restriction for the guiding function to a specific family of mappings. Second, the learned latent representation is committed to a specific

instance of the parameters.

Non-parametric Guidance

Snoek, Adams and Larochelle (2012) show a method to non-parametrically guide the learning of the latent codes. They address the limitations of the work of Bengio et al. (2007) by replacing the parametric classifier with a non-parametric model. Specifically, they connect a GPLVM to the autoencoder so that the latent space of GPLVM corresponds to the latent space of the autoencoder. They then train this hybrid model using a combined objective function which includes both the objective of the autoencoder and the GPLVM.

Limitations

The non-parametrically-guided autoencoder (NPGA) is conceived to learn good latent representations to improve discriminative tasks, that is, learning latent codes to achieve increased predictability of label information. They use label information (supervision) to guide a latent space learning process for an autoencoder; this is not a purely unsupervised learning task (thus that this method cannot be used when we do not have label information available). Furthermore, the NPGA does not propagate any useful uncertainty information from the latent space to the output space; this is due to the use of a determinist feed-forward network to the output.

3.3 Variational Autoencoder

Kingma and Welling (2013) introduced a variational method to perform efficient approximate learning and inference in directed probabilistic models with latent variables that have intractable posterior distributions.

More formally, given a training dataset $\mathcal{D} = \{\boldsymbol{x}_n\}_{n=1}^N$, with datapoints being independent and identically distributed, each datapoint $\boldsymbol{x}_n$ is generated by some process involving a corresponding latent code $\boldsymbol{z}_n$, *i.e.*, a vector $\boldsymbol{z}_n$ is generated by sampling from a prior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and then $\boldsymbol{x}_n$ is generated sampling from the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. This is effectively a probabilistic *decoder* because given a code $\boldsymbol{z}_n$ as input,

a datapoint $\boldsymbol{x}_n$ is generated. The set of parameters for the prior and decoder distributions is collectively represented by $\boldsymbol{\theta}$. The recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ is a probabilistic *encoder* parametrised by $\boldsymbol{\phi}$. Importantly, the following integral is usually intractable:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z})\, d\boldsymbol{z} \tag{3.3.1}$$

This implies that the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})$ is also intractable:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z})}{\int p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z})\, d\boldsymbol{z}} \tag{3.3.2}$$

The encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ is a variational distribution which approximates the (intractable) posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})$.

Variational Learning

The following is the variational lower bound to the marginal log likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ for a single datapoint $\boldsymbol{x}_n$ (Kingma and Welling, 2013):

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi} \mid \boldsymbol{x}_n) = -KL(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}_n) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_n)}\Big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n \mid \boldsymbol{z})\Big]\,, \tag{3.3.3}$$

where KL is the Kullback–Leibler divergence.

An estimator of this lower bound (Eq. 3.3.3) can be constructed using mini-batches of randomly drawn datapoints from the dataset $\mathcal{D}$ (the full derivation of this can be found in the work of Kingma and Welling (2013)):

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi} \mid \mathcal{D}) \approx \tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi} \mid \mathcal{D}^M) = \frac{N}{M} \sum_{i=1}^{M} \tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi} \mid \boldsymbol{x}_i)\,, \tag{3.3.4}$$

here, the mini-batch $\mathcal{D}^M = \{\boldsymbol{x}_i\}_{i=1}^M$ denotes a set of M random samples from the full dataset $\mathcal{D}$ of N datapoints.

The *variational autoencoder* (VAE) (Kingma and Welling, 2013) is a specific model that is trained by optimising this estimator of the marginal likelihood lower bound (Eq. 3.3.4).

The VAE assumes a multivariate Gaussian prior with zero mean and identity covariance matrix over the latent space: $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid \mathbf{0}, \boldsymbol{I})$.

Also, $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ is taken to be a multivariate Gaussian (or Bernoulli in the case of binary data) and the variational approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ is a multivariate Gaussian with diagonal covariance. The decoder distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ is implemented with a neural network, that is, the distribution parameters (such as Bernoulli parameters, or mean and standard deviation if the distribution is Gaussian) are computed from $\boldsymbol{z}_n$ as the output of a fully-connected neural network. Similarly, for the encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{\mu}_n, \boldsymbol{\sigma}_n^2 \boldsymbol{I})$ the parameters $\boldsymbol{\mu}_n$ and $\boldsymbol{\sigma}_n$ are the result of an encoding neural network, in other words, they are non-linear functions of datapoint $\boldsymbol{x}_n$ and the variational parameters $\boldsymbol{\phi}$ (the weights and biases of the neural network).

Reparametrisation Trick

We note that (as this is also very important for our model whose details we provide later in this thesis), it is often possible to rewrite a random variable as a deterministic function of a simpler random variable. For instance, the following Gaussian random variable:

$$\boldsymbol{z} = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \boldsymbol{I}) \ , \tag{3.3.5}$$

can be expressed as a deterministic function of $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$:

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \ , \tag{3.3.6}$$

(here, the symbol $\odot$ indicates an element-wise product).

Using the reparametrisation trick, for example, $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}_n)$ can be rewritten as in Eq. 3.3.6. Importantly, this expression is deterministic and differentiable with respect to the parameters, allowing the estimator of the lower bound (of Eq. 3.3.3) to be differentiable with respect to the network parameters.

Limitations

The VAE performs a variational approximation of a generative model by computing distribution parameters through feed-forward neural networks. It uses neural networks to encode the variational parameters (in a similar

fashion to Lawrence and Quiñonero-Candela (2006)). While this model provides a generative mapping, the feed-forward (decoder) network fails to propagate uncertainty from the latent space. Furthermore, another drawback of the model is the assumption of a specific Gaussian prior over the latent variables, which limits the form of the learned latent space. The independent Gaussian prior on the latent space does not promote a smooth manifold; any smoothness that does exist arises as a byproduct of the encoding neural network. This characteristic depends on the network architecture and is not directly parametrised. The key limitation of the VAE for our purposes is the lack of uncertainty propagation that results in poor performance with limited training data.

3.4 Generative Adversarial Networks

Another prominent generative model in unsupervised learning is the Generative Adversarial Network (GAN) (Goodfellow et al., 2014). The essential idea of this method is learning a distribution of the data implicitly by using two neural networks which *compete* with each other: a generator network G which transforms a noise variable $\boldsymbol{z}$ into a generated datapoint, and a discriminator network D which is used to discriminate between data from the generator's distribution and real training data.

More formally, in order to learn the distribution of the generator over data $\boldsymbol{x}$, a distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$ is defined on the input noise variables. The generative mapping with parameters $\boldsymbol{\theta}_g$ between $\boldsymbol{z}$ and the data space is represented as $G(\boldsymbol{z}; \boldsymbol{\theta}_g)$, it must be differentiable and it is implemented with a neural network. Similarly, the discriminative mapping $D(\boldsymbol{x}; \boldsymbol{\theta}_d)$ with parameters $\boldsymbol{\theta}_d$ is another neural network which takes as input a datapoint $\boldsymbol{x}$ and outputs a single scalar indicating the probability that $\boldsymbol{x}$ came from the dataset rather than the generator's distribution.

D is trained to maximise the probability of assigning the correct label to both real samples from the dataset and generated ones, G instead is trained to minimise the expression $\log(1 - D(G(\boldsymbol{z}; \boldsymbol{\theta}_g); \boldsymbol{\theta}_d))$. This corresponds for D and G to compete against each other in a minimax

game with value function $V(D, G)$:

$$\begin{aligned}\min_G \max_D V(D,G) = &\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log(D(\boldsymbol{x};\boldsymbol{\theta}_d))] \\ &+ \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z};\boldsymbol{\theta}_g);\boldsymbol{\theta}_d))]\end{aligned} \tag{3.4.1}$$

InfoGAN

One issue with the GAN is that it does not provide a smooth latent manifold for synthesis neither uncertainty in its estimates. From the plethora of different variants of the GAN model available in the literature we have chosen to include in our comparisons the InfoGAN (Chen et al., 2016), because with this method the authors aim to learn interpretable latent representations (this is achieved by maximising the mutual information between a subset of the GAN's noise variables and observations).

The standard GAN model does not impose any restriction on the way the input noise vector $\boldsymbol{z}$ is used by the generator. As a consequence, in order for the model to generate, it is very likely that the noise is used in a highly entangled manner. This also implies that there is no correspondence between semantic features of the data and the individual dimensions of $\boldsymbol{z}$. Chen et al. (2016) proposed to separate the input noise variables into two components: the vector $\boldsymbol{z}$, which is simply random noise sampled from the prior noise distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$, and the second part, a *latent code* $\boldsymbol{c}$, to target the structured semantic features of the data. So, differently from the GAN, in the InfoGAN model the generator is $G(\boldsymbol{z}, \boldsymbol{c}; \boldsymbol{\theta}_g)$, taking as input both the unstructured noise $\boldsymbol{z}$ and latent code $\boldsymbol{c}$.

However, since the generator can lead to trivial solutions of the objective ignoring the latent code $\boldsymbol{c}$, Chen et al. (2016) add an *information-theoretic regularisation* $I(\boldsymbol{c}; G(\boldsymbol{z}, \boldsymbol{c}; \boldsymbol{\theta}_g))$ to the standard GAN's objective of Eq. 3.4.1, leading to the following new minimax objective:

$$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(\boldsymbol{c}; G(\boldsymbol{z}, \boldsymbol{c}; \boldsymbol{\theta}_g)) \ , \tag{3.4.2}$$

where I indicates the mutual information and λ is a hyperparameter. The term $I(\boldsymbol{c}; G(\boldsymbol{z}, \boldsymbol{c}; \boldsymbol{\theta}_g))$ in the objective encourages the mutual information between the generator distribution and the latent codes $\boldsymbol{c}$ to be high; in

other words, this means that the latent codes should be *informative* with respect to the generated samples.

One issue with the minimax objective of Eq. 3.4.2 is that the mutual information component is difficult to maximise directly as access to the posterior $p(\boldsymbol{c} \mid \boldsymbol{x})$ is needed. For this reason Chen et al. (2016) define a variational lower bound which uses an approximate distribution $Q(\boldsymbol{c} \mid \boldsymbol{x})$ (diagonal Gaussian distribution) for the posterior $p(\boldsymbol{c} \mid \boldsymbol{x})$. Thus, the InfoGAN objective with the variational regularisation of the mutual information term becomes:

$$\min_{G,Q} \max_{D} V_{\text{InfoGAN}}(D, G, Q) = V(D, G) - \lambda L_I(G, Q) \ . \tag{3.4.3}$$

In the above Equation (3.4.3) the variational term is defined as follow:

$$L_I(G, Q) = \mathbb{E}_{\boldsymbol{c}\sim p(\boldsymbol{c}), \boldsymbol{x}\sim G(\boldsymbol{z},\boldsymbol{c};\boldsymbol{\theta}_g)}[\log Q(\boldsymbol{c} \mid \boldsymbol{x})] + H(\boldsymbol{c}) \ , \tag{3.4.4}$$

where $H(\boldsymbol{c})$ is treated as a constant, and Q is parametrised as a neural network.

Limitations

Although the InfoGAN represents an improvement over the standard GAN in terms of learning a better latent space to generate from (by imposing additional structure on this space), the main limitation is that, similarly to the VAE, the latent space does not provide any uncertainty information, furthermore, the generator (a neural network) is a deterministic function of the unstructured noise $\boldsymbol{z}$ and latent code $\boldsymbol{c}$, so the InfoGAN suffers of the lack of uncertainty propagation throughout the model.

3.5 Shape Boltzmann Machine

The Shape Boltzmann Machine (SBM) (Eslami et al., 2014) is a specific architecture of the Boltzmann machine. It consists of three layers: a rectangular layer of $N \times M$ visible units $\boldsymbol{v}$, and two layers of latent variables: $\boldsymbol{h}^1$ and $\boldsymbol{h}^2$. Each hidden unit in $\boldsymbol{h}^1$ is connected only to one of the four subsets of visible units of $\boldsymbol{v}$ (Fig. 3.2). Each subset forms a

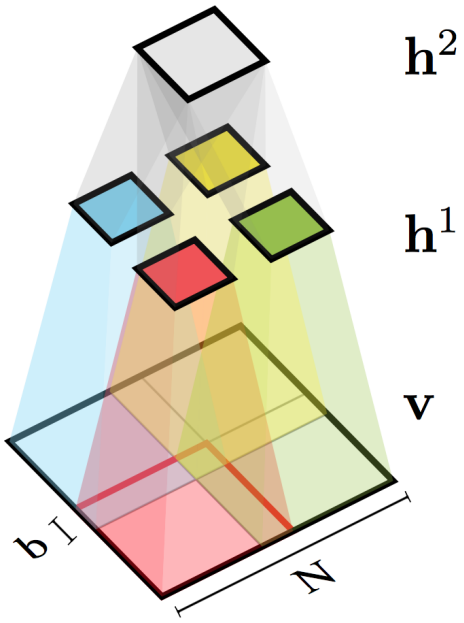


Figure 3.2: Graphical model of the SBM. (The figure is taken from Eslami et al. (2014).)

rectangular patch and the weights of each patch (except the biases) are shared so that a patch effectively behaves as a local receptive field. To avoid boundary inconsistencies, the patches are slightly overlapped (in Fig. 3.2, the overlap has size b). Layer $\boldsymbol{h}^2$ is fully connected to $\boldsymbol{h}^1$.

While layer $\boldsymbol{h}^1$ captures local low-level features in the images, layer $\boldsymbol{h}^2$, with its full connectivity, enforces global constraints and captures higher-order dependencies between visible units belonging to different patches. Furthermore, $\boldsymbol{h}^2$ has fewer units compared to $\boldsymbol{h}^1$. This restriction helps avoiding overfitting and improves generalisation.

Limitations

While the SBM offers improved generalisation over a DBN with the same number of parameters, the SBM has a fixed structure which is not easily extended to more layers or patches. In contrast, a DBN, as a stack of simple RBMs, has a more generic and flexible structure which can be adapted easily and combined with other models. Furthermore, like the DBN, the SBM lacks a proper generative process from the latent space.

3.6 ShapeOdds

In *ShapeOdds* (Elhabian and Whitaker, 2017) an observed high-dimensional silhouette space is assumed to be governed by an underlying latent low-dimensional process. The model specifies a detailed latent variable structure including a Gaussian Markov random field with individual Bernoulli random variables for the pixel lattice, that is, a spatially correlated field of Bernoulli random variables which corresponds to the shape space and is controlled by a parameter map.

In order to describe the model more formally let us consider a spatial domain $\Omega \subset \mathbb{R}^D$ containing D pixels x. Each pixel in a silhouette $\boldsymbol{f} \in \{0,1\}^D$ is either 0 or 1 depending on whether the pixel belongs to the background or to the silhouette. Given the shape space $\mathcal{F}$ and a silhouette dataset $F = \{\boldsymbol{f}_n\}_{n=1}^N \subset \mathcal{F}$, the unknown shape distribution is $p(\boldsymbol{f})$. In the ShapeOdds latent variable model, this distribution is assumed to be governed by a low-dimensional shape generative process that uses L independent latent variables $\boldsymbol{z} \in \mathbb{R}^L$ with $L \ll D$.

ShapeOdds Generative Process

Elhabian and Whitaker (2017) define the ShapeOdds generative process, with model parameters $\boldsymbol{\Theta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{W}, \boldsymbol{w}_0\}$ and priors' hyperparameters $\boldsymbol{\Psi} = \{\boldsymbol{\lambda}, \boldsymbol{\beta}\}$ as follows:

$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \ , \tag{3.6.1}$$

$$p(\boldsymbol{f}_n \mid \boldsymbol{z}_n, \boldsymbol{\Theta}) = \text{Expon}(\boldsymbol{\phi}_n)\text{Mrf}(\upsilon) \ , \tag{3.6.2}$$

$$\text{Expon}(\boldsymbol{\phi}_n) = \prod_{x \in \Omega} \text{Expon}(\phi_n(x)) \ , \tag{3.6.3}$$

$$\boldsymbol{\phi}_n = \boldsymbol{W}\boldsymbol{z}_n + \boldsymbol{w}_0 \ , \tag{3.6.4}$$

$$p(\boldsymbol{w}_0 \mid \lambda_0) \sim \text{GMrf}(\lambda_0) \ , \tag{3.6.5}$$

$$p(\boldsymbol{w}_l \mid \lambda_l, \beta_l) \sim \text{GMrf}(\lambda_l)\text{Ard}(\beta_l) \ , \tag{3.6.6}$$

with:

$$\text{Expon}(\phi(x)) = \exp(f(x)\phi(x) - \log(1 + e^{\phi(x)})) \ , \tag{3.6.7}$$

$$\text{GMrf}(\lambda) = \mathcal{N}(\boldsymbol{0}_D, \lambda^{-1}\boldsymbol{S}) \ , \tag{3.6.8}$$

In the above equations x indicates a pixel value. In Eq. 3.6.2, the matrix $\boldsymbol{S}$ encodes the structure of the Gaussian Markov random field prior, this has hyperparameter $\upsilon > 0$ and encodes the spatial regularity of a silhouette. Eq. 3.6.3 encodes the axiom of conditional independence of the observed variables given the latent variables. The ARD (automatic relevance determination) prior in Eq. 3.6.6 is a prior to further regularise the solution space, that is, a zero-mean isotropic Gaussian on $\{\boldsymbol{w}_l\}_{l=1}^{L}$ parametrised by β_l.

Learning

Training the ShapeOdds model is not straightforward; unfortunately, the fact that the Gaussian prior is not conjugate to the Bernoulli likelihood implies the computation of an intractable logistic-Gaussian integral. Moreover, a simple point estimate such as maximum likelihood for the posterior would ignore precious probabilistic uncertainties and would lead to overfitting. In order to train the model, a variational approximation to the marginal likelihood is used instead, resulting in a tractable expectation-maximisation (EM) procedure, retaining the benefits of posterior uncertainty.

Assuming the following posterior variational distribution:

$$q(\boldsymbol{z}_n \mid \boldsymbol{\gamma}_n) = \mathcal{N}(\boldsymbol{z}_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n) \ , \tag{3.6.9}$$

with mean and covariance parameters collectively indicated as $\boldsymbol{\gamma}_n = \{\boldsymbol{m}_n \in \mathbb{R}^L, \boldsymbol{V}_n \in \mathbb{R}^{L\times L}\}$, a variational lower bound to the log-marginal likelihood can be obtained using Jensen's inequality:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\Theta}) \geq \mathcal{L}^J(\boldsymbol{\Theta}, \boldsymbol{\gamma}) = \sum_{n=1}^{N} \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{\gamma}_n)}\Big[\log \frac{p(\boldsymbol{z}_n \mid \boldsymbol{\Theta})}{q(\boldsymbol{z}_n \mid \boldsymbol{\gamma}_n)}\Big] + \\ + \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{\gamma}_n)}\Big[\log p(\boldsymbol{f}_n \mid \boldsymbol{z}_n, \boldsymbol{\Theta})\Big] \ . \end{aligned} \tag{3.6.10}$$

In Eq. 3.6.10 the first term is a Kullback-Leibler divergence which encourages the variational distribution $q(\boldsymbol{z}_n \mid \boldsymbol{\gamma}_n)$ to approximate the prior $p(\boldsymbol{z}_n \mid \boldsymbol{\Theta})$. The equation can be further expanded and simplified and then optimised using a variational EM procedure. We omit the mathematical details here and instead refer the reader to the paper of Elhabian and

Whitaker (2017) for a more formal treatment.

Limitations

Thanks to its Bernoulli lattice and Markov random field prior, the ShapeOdds model confers state-of-the-art performance to generative shape modelling and captures many of the desired properties, including a generative probabilistic model that propagates uncertainty. However, ShapeOdds does not explicitly model any smoothness in the latent space. Furthermore, the approach taken is quite different to the VAE and GAN models; Elhabian and Whitaker (2017) define a very detailed probabilistic model, in contrast, we argue that a model should be more flexible in this respect, allowing the structure to be learned from the data directly and ensuring that uncertainty quantification is still maintained throughout. The explaining-away property of directed probabilistic models (such as ShapeOdds) comes at the price of losing flexibility and abstraction typical of deep models (*e.g.*, deep belief networks) which do not assume any domain knowledge of the modelling problem at hand.

3.7 GPLVM Representations

A possible workaround to the problem of non-Gaussian likelihoods is to perform a deterministic transformation to a domain where the data is approximately Gaussian. This has been successful for domains where, for example, the shape can be represented in a new geometric representation away from pixels, *e.g.*, parametric curves (Campbell and Kautz, 2014; Prisacariu and Reid, 2011). However, this is application dependent and not suitable for arbitrary pixel based silhouettes considered here. A common approach that retains the pixel grid is to transform it into a level-set problem via the distance transform (*e.g.*, Prisacariu and Reid (2012)).

Distance Transforms

For binary images, for example, the Euclidean distance transform, replaces each foreground pixel (value 1) with its Euclidean distance to the nearest

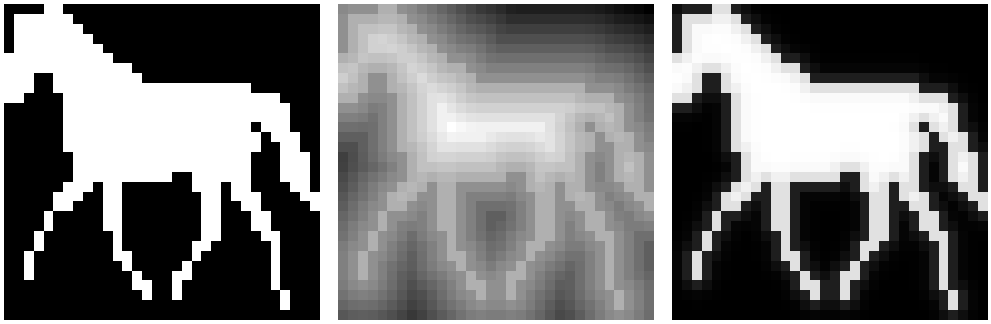
background pixel (value 0). The background pixels are left to their value of 0. More formally, the Euclidean distance transform calculates each new pixel value t as follows:

$$t = \sqrt{(f_x - b_x)^2 + (f_y - b_y)^2} \ , \tag{3.7.1}$$

where f_x and f_y are the coordinates of a foreground pixel in the input image and b_x and b_y the coordinates of its nearest background pixel.

The *signed* version of the Euclidean distance transform also replaces each background pixel by the negative value of its Euclidean distance from its nearest foreground pixel. To transform a binary image using the signed Euclidean distance transform we can simply invert the binary image (that is, we swap each pixel value from 0 to 1 and vice versa), we then apply the Euclidean distance transform and subtract this to the Euclidean distance transform of the original (non-inverted) binary image. Given a transformed image (using the signed Euclidean distance transform), the corresponding binary image can be recovered by simply setting a threshold at 0.

The signed Euclidean distance transform converts the binary image data in a way that makes it more suitable for the Gaussian likelihood assumption of the GPLVM. After training a GPLVM on transformed data, predictions from the model can then be untransformed either by thresholding at 0 or by using the hyperbolic tangent function. The hyperbolic tangent is differentiable, which is an advantage at test time for some evaluation algorithms. The hyperbolic tangent, however, "softly" thresholds the input by squishing each pixel value to the continuous range $[-1, 1]$ (this is then normalised between $[0, 1]$ to match the range of the binary data), therefore, the output is a "smoothed" version of the more correct binary image that would be obtained by thresholding at 0 with a step function. Fig. 3.3 shows an example of the signed Euclidean distance transform applied to a binary image of a horse and the corresponding untransformed output.



(a) Binary (b) Transformed (c) Untransformed

Figure 3.3: (a) Original input binary image. (b) Output of the signed Euclidean distance transform. (c) Untransformed image using the sigmoid function.

Limitations

The use of a distance transform can improve results in some settings, however, the main drawback is that the uncertainty is not correctly preserved because of the necessary thresholding process and therefore it is not correctly captured in predictions.

In our comparisons in Chapter 5, we denote this model as *GPLVMDT*, this is trained on pre-transformed data using the signed Euclidean distance, and the sigmoid function is used to untransform predictions.

3.8 Summary of the Reviewed Methods

We conclude this background chapter by providing below, in Table 3.2, a detailed summary of the good properties and issues of the methods that we have reviewed so far in this work.

We note that there is a plethora of newer variants and extensions of the VAE and GAN models. Comparing against all of them is not feasible and out of the scope of this thesis. We have included the standard VAE in our comparisons because it is a very popular model and it provides a latent space from which we can generate samples easily. The InfoGAN is also a popular generative model and in addition to the standard GAN it also aims to learn interpretable latent representations.

Table 3.2: Summary of the advantages and disadvantages of the reviewed methods.

Method	Advantages	Disadvantages
NPGA	Non-parametric guidance to the latent space.	Requirement of labelled data. Aim of learning representations for better label predictability (for this reason we do not use this model in our comparisons later in this thesis).
GPLVM	The model ensures a smooth mapping from the latent space to the observed data, with principled uncertainty propagation.	Not suitable to model image data because of the Gaussian likelihood assumption.
GPLVMDT	Like the GPLVM it provides a smooth mapping from latent space to data space. The model represents an improvement over the GPVLM in terms of modelling image data thanks to the use of the distance transform.	Because of the use of the distance transform, uncertainty is not propagated correctly through the model.
DBN	Flexible stochastic neural network structure that learns hierarchical representations of the data.	No directed generative sampling process from a well-defined latent representation. No explicit representation of the uncertainty for the latent points. Uncertainty arises implicitly through the propagation of samples (point estimates) at each layer. Latent dimensions are independent.

Continued on next page

Table 3.2 – continued from previous page

Method	Advantages	Disadvantages
SBM	Shared weight architecture for better shape modelling (compared to a normal DBN).	It is not clear whether the model can be easily scaled up to more than two layers of hidden units. The same problems of the DBN are present in the SBM as well.
VAE	The model provides a simple generative process using a decoder network.	Assumption of an isotropic multivariate Gaussian prior with zero mean over the latent space. The network mappings for the decoder and encoder are deterministic. So, there is no explicit uncertainty propagation from the latent space. No explicit smoothness modelling for the latent space.
InfoGAN	The model provides a simple generative process using a generator network.	No uncertainty propagation because generator and discriminator networks are deterministic. Smoothness is not modelled explicitly.
ShapeOdds	Good probabilistic generative shape model thanks to the individual Bernoulli variables and MRF prior capturing spatial correlations.	Highly detailed and complex model architecture (not as flexible as a deep network). No explicit smoothness modelling for the latent space.

Chapter 4

The GPDBN Model

In Chapter 2 we reviewed Gaussian processes and deep belief networks and their properties, in this chapter we present a method to combine the benefits of both into a single model (the GPDBN).

4.1 Overview of the Model

In Section 2.2.4 we have explained that the GPLVM (Lawrence, 2005) is a non-parametric model based on Gaussian processes and it uses them in an unsupervised way. It learns a low-dimensional representation of the data that is smooth thanks to the choice of a smooth covariance function (such as the squared exponential kernel), and provides predictive uncertainty for each location in the latent space. A smooth latent representation with uncertainty is desirable for generating data because smoothness guarantees that if two points are close together in the low-dimensional space then the corresponding high-dimensional samples will also be close to each other, in addition, predictive uncertainty gives us an indication about where to generate from in the low-dimensional space to obtain good samples. In simple terms, these properties are important because they confer good *interpretability* to the generating process. Although the GPLVM provides a smooth mapping with uncertainty, unfortunately, the model is not suitable for modelling image data because of the Gaussian likelihood assumption (Section 2.2.4).

Deep belief networks, on the other hand, are parametric models with stochastic units. Their deep hierarchical structure allows them to learn representations of the data by composing features learned by all the layers, this leads to efficient and flexible representations (lower layers learn simpler lower-level

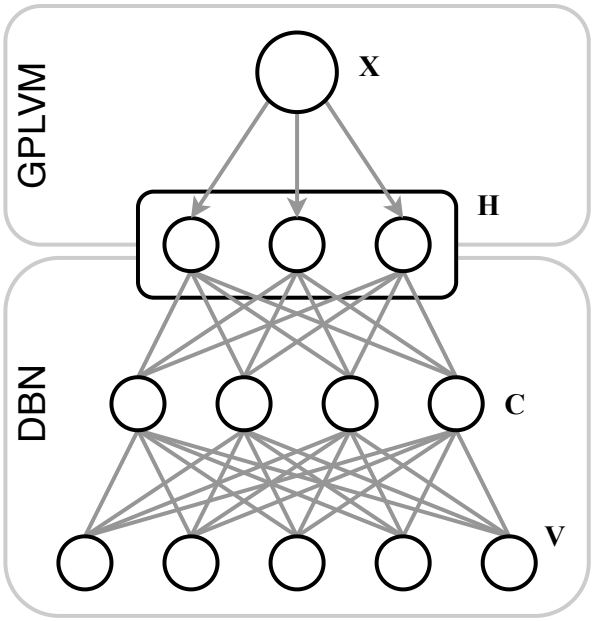


Figure 4.1: A graphical representation of the GPDBN model, where $\boldsymbol{X}$ represents the latent variables, $\boldsymbol{H}$ the Gaussian activations 4.4.1, $\boldsymbol{C}$ is a layer of Concrete units and $\boldsymbol{V}$ the observed (data) space.

features, higher layers learn more complex and abstract features). Deep belief networks unfortunately lack of a proper generative process where one can choose a location in the latent space and generate a high-dimensional sample from it (instead, conditioning on a datapoint and performing Gibbs sampling is the standard way to generate samples). The learned latent space is not guaranteed to be smooth and no predictive uncertainty is provided for the generated output (Section 2.1.3).

We want to merge the benefits of a smooth latent space with principled uncertainty propagation and efficient deep learning representations into a single model. The key idea of our GPDBN model is to connect the DBN and GPLVM so that the output space of the GPLVM corresponds the latent space of the DBN (Fig. 4.1). Unfortunately, because of the parametric and non-parmetric nature of the two models respectively, connecting them into a single model is not straightforward. If a GPLVM is simply stacked on top of a DBN, uncertainty propagation throughout the joint model is not preserved. Most importantly, while a DBN (with standard Bernoulli units, which are non-differentiable with respect to the model parameters) is trained with contrastive divergence (2.1.2), the GPLVM instead is trained by optimising an objective function.

In the next sections we describe in more detail how the benefits of the

GPLVM and DBN can be combined into a single model and propose a method that allows the model to be optimised efficiently by minimising a single objective function, while also preserving uncertainty propagation.

4.2 Concrete Layers

An important limitation of standard deep belief networks is the non-differentiability of binary (Bernoulli) units with respect to the parameters. To overcome this difficulty there are ways to approximate discrete random variables with continuous random variables. Building on the work of Maddison, Mnih and Teh (2017) on the *Concrete* random variables (*i.e.*, continuous relaxations of discrete random variables), we construct a DBN where all the layers use Concrete relaxations to binary units (except for the uppermost hidden layer, as we explain later in this section). This allows us to draw samples, in an analogous manner to the reparametrisation trick (Sec. 3.3), using a function that is differentiable with respect to the model parameters. Specifically, we use the following function as a Concrete relaxation to the Bernoulli random variable:

$$\text{Con}\,(p, u) = \text{Sigmoid}\left[\frac{1}{\lambda}(\log p - \log(1-p) + \log u - \log(1-u))\right] \quad , \quad (4.2.1)$$

where p is the Bernoulli probability parameter, λ is a scaling factor, which we fix to 0.1 for a good approximation, and u is a uniform sample from $[0, 1]$. In Eq. 4.2.1 the random variable u does not depend on the probability parameter p, this means that the functional relation between u and the output sample is differentiable with respect to p. In Fig. 4.2 the functional relation between the input u and the corresponding output value of the sample is shown for a Concrete variable and a Bernoulli variable.

The uppermost hidden layer of the DBN that we construct for our model does not have Concrete units, instead it has Gaussian units, this is to interface with the Gaussian likelihood of the GPLVM (Eq. 2.2.28) that we connect to this DBN.

4.3 Generative Process

Generating from the model is very simple. Eq. 4.3.2 specifies the full generative process. A sample $\boldsymbol{s}$ from the model is drawn by first generating a D-dimensional

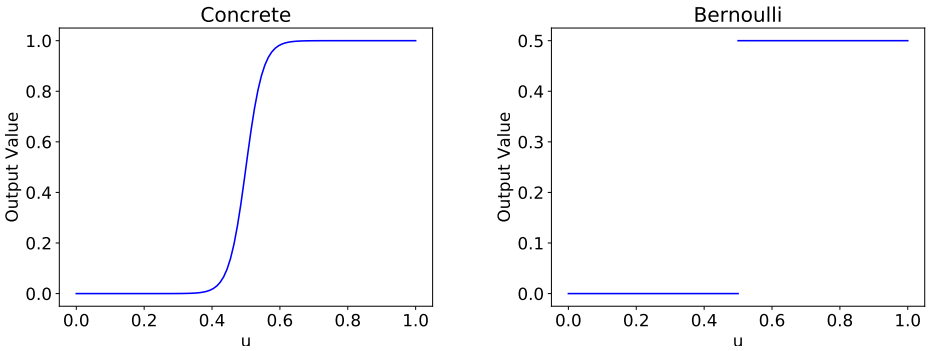


Figure 4.2: These plots show the functional relations between an input sample $u \sim \text{Uniform}[0, 1]$ and corresponding value from a Concrete relaxation to a Bernoulli variable with $\lambda = 0.1$ (on the left) and a standard Bernoulli random variable (on the right). The function is sigmoidal for the Concrete variable and approximates the step function of the Bernoulli variable, which presents a discontinuity and is non-differentiable. (In both plots the distribution parameter p is fixed to 0.5).

hidden sample $\boldsymbol{h}$ from latent location $\boldsymbol{x}$ as follows:

$$\boldsymbol{h}(\boldsymbol{x}) = (\boldsymbol{m}^{\text{GP}} + \sigma^{\text{GP}} \times \boldsymbol{\epsilon}) \odot \boldsymbol{\sigma}^{\text{DBN}} + \boldsymbol{h}_{\mu} \ , \tag{4.3.1}$$

using $\boldsymbol{m}^{\text{GP}}$ and σ^{GP} as the predictive mean and standard deviation of the GPLVM given latent point $\boldsymbol{x}$. This is combined with a sample $\boldsymbol{\epsilon}$, a $1 \times D$ vector of spherical Gaussian noise (this is an instance of the reparametrisation trick, allowing the expression to be differentiated with respect to the model parameters). The term $\boldsymbol{\sigma}^{\text{DBN}}$ is the $1 \times D$ vector of standard deviation parameters of the Gaussian units; $\boldsymbol{h}_{\mu}$ is the mean vector that is subtracted from $\boldsymbol{H}$ in the normalisation step at training time (as explained in Sec. 4.4). The sample $\boldsymbol{h}$ is then propagated down through the DBN, sampling layer-by-layer, to give an output high-dimensional sample $\boldsymbol{s}$ (this has the time complexity of a simple forward pass of a stochastic network). More formally, Eq. 4.3.2 defines the full generative process to obtain a sample $\boldsymbol{s}$ (with DBN indicating the network layers):

$$\boldsymbol{s} = \text{DBN}(\boldsymbol{h}(\boldsymbol{x})) \ . \tag{4.3.2}$$

4.4 Gaussian Activation Matrix

By generating from a point in the latent space, the GPLVM returns as output a single scalar representing the predictive uncertainty σ_n^{GP} as well as a corre-

sponding D-dimensional vector (with D being the size of the output space for the GPLVM). To join the two models and propagating uncertainty at training time we define a matrix of activations from the Gaussian units which we call $\boldsymbol{H}$ (this has size $N \times D$ and is used in the training objective of Eq. 4.5.1):

$$\boldsymbol{H} = \boldsymbol{A} + \boldsymbol{\sigma}^{\mathrm{GP}} \otimes \boldsymbol{\sigma}^{\mathrm{DBN}} \odot \boldsymbol{\mathcal{E}} \ , \tag{4.4.1}$$

where $\boldsymbol{A} = [\boldsymbol{m}_1, \ldots, \boldsymbol{m}_N]^\top$ is a matrix in which each row is the mean output of the Gaussian units corresponding to each input training datapoint (N in total). This is combined with $\boldsymbol{\sigma}^{\mathrm{GP}}$, the $N \times 1$ vector of predictive standard deviations from the GPLVM, and $\boldsymbol{\sigma}^{\mathrm{DBN}}$, the $1 \times D$ vector of standard deviation parameters of the Gaussian units. (The symbol $\otimes$ specifies an outer product, and $\odot$ specifies an element-wise product.)

The $\boldsymbol{H}$ matrix represents the observed data for the GPLVM and is updated at each training iteration by sampling $\boldsymbol{\mathcal{E}}$, an $N \times D$ matrix of independent Gaussian noise, *i.e.*, each individual $\mathcal{E}_{n,d}$ is sampled as follows: $\mathcal{E}_{n,d} \sim \mathcal{N}(0, 1)$.

Importantly, this is a second application of the reparametrisation trick. In Eq. 4.4.1, none of $\boldsymbol{A}$, $\boldsymbol{\sigma}^{\mathrm{GP}}$ and $\boldsymbol{\sigma}^{\mathrm{DBN}}$ depend on the noise $\boldsymbol{\mathcal{E}}$ (*i.e.*, they are not functions of $\boldsymbol{\mathcal{E}}$) which makes the expression differentiable with respect to the model parameters.

At each iteration, $\boldsymbol{H}$ is always normalised, to match our zero mean Gaussian process assumption, by subtracting its column-wise mean and dividing by $\boldsymbol{\sigma}^{\mathrm{DBN}}$.

In the next section we provide the details of how the GPDBN is trained.

4.5 Objective Function

We propose to train the model using an objective function that consists of terms that are in contrast with each other. A *data* term to ensure that the observed data is well represented by the model (a cross entropy term between the training data and the generated samples from the model). A *complexity* term to encourage a simple (low complexity) latent space $\boldsymbol{X}$ through its covariance matrix $\boldsymbol{K}$ to prevent overfitting. A *joint* term to "glue" the two models (GPLVM and DBN) together by ensuring that the covariance matrix $\boldsymbol{K}$ is a good model of the covariance of the Gaussian units at the top of the DBN. This in turn, ensures that the DBN learns an appropriate network to give sensible Gaussian activations rather than the unconstrained binary activations from a normal DBN. A final term to encode a *prior* (squared L2-Norm) on $\boldsymbol{X}$

that encourages the latent points to stay close to the origin.

Thus, given a dataset $\mathcal{D} = \{\boldsymbol{t}_n\}_{n=1}^N$, we train the GPDBN model end-to-end by minimising the following objective function jointly with respect to all the parameters of the model (that is, the parameters of the network, the covariance function's hyperparameters and the matrix of latent points $\boldsymbol{X}$. For simplicity we omit these from the notation to avoid clutter):

$$L = \underbrace{\sum_{n=1}^{N} [\boldsymbol{t}_n \log(\boldsymbol{s}_n) + (1 - \boldsymbol{t}_n) \log(1 - \boldsymbol{s}_n)]}_{\text{data term}} + \\ + \underbrace{\frac{1}{2} \text{Tr}\left[\boldsymbol{K}^{-1} \boldsymbol{H} \boldsymbol{H}^\top\right]}_{\substack{\text{joint} \\ \text{term}}} + \underbrace{\frac{D}{2} \log |\boldsymbol{K}|}_{\substack{\text{complexity} \\ \text{term}}} + \underbrace{||\boldsymbol{X}||^2}_{\substack{\text{prior} \\ \text{term}}} . \tag{4.5.1}$$

In Eq. 4.5.1, $\boldsymbol{t}_n$ is a training datapoint, $\boldsymbol{s}_n$ is a sample from the model (which is obtained as we explain later in Sec. 4.3). The covariance matrix $\boldsymbol{K}$ of the latent points $\boldsymbol{X}$ is given by:

$$\boldsymbol{K} = k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I} . \tag{4.5.2}$$

The variance of the noise parameter is specified by σ^2 (with $\boldsymbol{I}$ an $N \times N$ identity matrix), D is the number of Gaussian units in the uppermost DBN layer (equal to the dimension of the GPLVM output space).

The application of the reparametrisation trick ensures that samples can be taken in an efficient manner during training with gradients propagated throughout all parts of the network. The use of sampling and stochastic networks allows uncertainty to be propagated down through the entire model as well to ensure uncertainty is well quantified both at training and test time.

The single objective function proposed in Eq. 4.5.1 allows us to train the model using standard backpropagation, thus layer-by-layer contrastive divergence training is not required. The training time complexity of the GPDBN model is the same as for a normal feed-forward neural network. Optionally, the DBN can be pre-trained for a few iterations before training the full model with backpropagation, this provides a good initialisation for the weights of the network and might result in a better learned latent space (we note that, however, this DBN pre-training step in not essential for the model and does not lead to significant differences in terms of performance in most cases. In our experiments in Chapter 5 we do not use any pre-training, except

for where explicitly mentioned).

The application of the principal component analysis (PCA) dimensionality reduction method on the training data provides a good initialisation for the matrix of latent points $\boldsymbol{X}$ (this is also a very common way to initialise $\boldsymbol{X}$ in the standard GPLVM).

4.6 Mini-batching in Training

The objective 4.5.1 can be evaluated on an uniformly drawn subset of training data $\{\boldsymbol{t}_b\}_{b=1}^{B}$ yielding an estimator for the full objective,

$$\begin{aligned} L_{\text{batched}} \simeq \frac{N}{B}\sum_{b=1}^{B}(\boldsymbol{t}_b \log(\boldsymbol{s}_b) + (1-\boldsymbol{t}_b)\log(1-\boldsymbol{s}_b)) + \\ + \frac{N}{2B}\text{Tr}\left[\boldsymbol{K}_B^{-1}\boldsymbol{H}_B\boldsymbol{H}_B^{\top}\right] + \frac{ND}{2B}\log|\boldsymbol{K}_B| + \frac{N}{B}||\boldsymbol{X}_B||^2 \ , \end{aligned} \tag{4.6.1}$$

where $\boldsymbol{H}_B$ and $\boldsymbol{K}_B$ corresponds to $\boldsymbol{H}$ and $\boldsymbol{K}$ evaluated on the subset $\boldsymbol{X}_B$ of $\boldsymbol{X}$. Using this estimator the model can be optimised using mini-batching to scale linearly to larger datasets. We note that the matrix inversion introduces some bias into the estimator; empirical results (in Sec. 5.7) suggest that this is small and removing it is a topic for future work.

4.7 Prediction and Projection

Since we have a simple sampling process, we can estimate uncertainty for our predictions by taking the empirical mean of a set of P samples from the model as

$$\bar{\boldsymbol{s}}_* = \frac{1}{P}\sum_{p=1}^{P}\boldsymbol{s}_p(\boldsymbol{h}_p(\boldsymbol{x}_*)) \ , \tag{4.7.1}$$

for any latent location $\boldsymbol{x}_*$.

Moreover, since we can efficiently take gradients through the sampling process, we can project new datapoints into the latent space by minimising the error between the datapoints and samples from the model with respect to the latent locations for predictions from a set of random starting locations in the manifold. More formally, we define the *projection* error as follows:

$$L_{\text{proj}}(\boldsymbol{x}_*) = \frac{1}{V}\sum_{i=1}^{V}(\boldsymbol{t}_* \log(\boldsymbol{s}_i) + (1-\boldsymbol{t}_*)\log(1-\boldsymbol{s}_i)) + \gamma \times \log(\sigma^2(\boldsymbol{x}_*)) \ . \tag{4.7.2}$$

The closest generated datapoint to a test datapoint $\boldsymbol{t}_*$ can be found by minimising the objective in Eq. 4.7.2 with respect to a latent location $\boldsymbol{x}_*$. The number of samples used to evaluate the cross entropy to the test datapoint is indicated by V. The log term in Eq. 4.7.2 is the log predictive variance of the latent location $\boldsymbol{x}_*$ (as defined in Eq. 2.2.23), this encourages the model to generate low-variance samples from the manifold. The scaling factor γ ensures that the two terms have approximatively the same scale.

4.8 Scaling via Convolutional Architecture

The fully-connected conditional independence of the RBM layers limits scalability in terms of input image size. This can be circumvented by adding convolution and deconvolution steps in the network layers to replace the dense matrix product (Eq. 2.1.2).

In simple terms, in two-dimensions, the operation of *convolving* a 2D *kernel* (*i.e.*, a small matrix that acts like a multiplicative filter for the input) corresponds to sliding the kernel across the input, so that at each location the products between the kernel values and the input values are taken and summed up to obtain the output value at the current location. A *transposed convolution* (or *deconvolution*) is simply a convolution in the opposite direction.

Adding convolutions and deconvolutions in the network layers of the model to replace the fully-connected layers allows a drastic reduction in the number of model parameters thanks to the fact that the kernel weights are reused at multiple locations in the input, in this way the model can handle larger input images.

Chapter 5

Experiments

In keeping with previous work, we evaluated our models in terms of four types of experiments: (i) *Synthesis*, that is, generating samples that are plausible. (ii) *Representation and Generalisation*, demonstrating the ability to capture the variability of the silhouettes away from the training data. (iii) *Smoothness*, evaluating the quality of the learned latent space through interpolation; smooth trajectories in the latent space should produce smooth variations in the silhouette space. (iv) *Scaling*, evaluating how the model performs with respect to the size of the training dataset.

5.1 Our Models

In the comparisons, our main model (which we refer to as GPDBN) has three layers. From the bottom (observed) to the top (hidden) layer the architecture consists of 200 (Concrete units), 100 (Concrete) and 50 (Gaussian). The connected GPLVM layer has only 2 latent dimensions for easy visualisation.

GPSBM

Our second model, which we call *GPSBM*, is similar to the GPDBN. Here, the three-layer network has been replaced with an SBM architecture of Eslami et al. (2014) with hidden Concrete units in the bottom layer and hidden Gaussian units at the top.

We have chosen to experiment with this variant of the GPDBN model by replacing the DBN with an SBM because the SBM provides a network architecture with shared weights which captures the structure of binary images better compared to a normal DBN.

We implemented all our models in the TensorFlow (Abadi et al., 2015) framework and optimised (4.5.1) directly using the Adam optimiser (Kingma and Ba, 2014).

5.2 Baselines

We compared our models to seven baselines: (i) A vanilla GPLVM with 2 latent dimensions. (ii) GPLVMDT, *i.e.*, a GPLVM operating on the signed Euclidean distance function representation (as described in Sec. 3.7). (iii) The ShapeOdds model (Elhabian and Whitaker, 2017). (iv) A DBN with binary units and the same architecture as our GPDBN. (v) The SBM (Eslami et al., 2014) model with binary units (trained layer by layer with contrastive divergence like the DBN) with the same architecture as our GPSBM. (vi) The VAE (Kingma and Welling, 2013) model with the same architecture as our GPDBN (mirrored for the decoder) and 2 latent dimensions. (vii) An InfoGAN (Chen et al., 2016) with same fully-connected network architecture as the VAE and GPDBN (mirrored for the discriminator) and 2 latent dimensions of structured noise.

We point out that there are many variants of the VAE and GAN models. It is not possible and out of the scope of this thesis to compare our models against all of the VAE and GAN variants. For our comparisons we simply chose the standard VAE, which provides a latent space that can be sampled easily, and the InfoGAN, which in addition to the standard GAN also aims to learn an interpretable latent space.

5.3 Datasets

In keeping with previous work, we trained the models on the Weizmann horse dataset (Borenstein, Sharon and Ullman, 2004), which consists of 328 binary silhouettes of horses facing left. The limited number of training samples and the high variability in the position of heads, tails, and legs make this dataset difficult. We also trained the models on 300 binary images from the Caltech101 dataset of motorbikes facing right (Fei-Fei, Fergus and Perona, 2004). All images in both datasets have been cropped and normalised to 32×32 pixels. The test datasets consisted of the challenging held-out data from Eslami et al. (2014), an additional 14 horses and 9 motorbikes not contained in the training datasets. We also used a version of MNIST digit dataset (LeCun, Cortes and Burges, 1998) corrupted with salt-and-pepper noise and a *star* dataset

(of which the details will be given later in this chapter) that was specifically designed to test the smoothness of the latent spaces of the compared models.

5.4 Synthesis

In Fig. 5.1 we show the manifold learned by the GPDBN on the Weizmann horse dataset. Each blue point on the manifold represents the latent location corresponding to a training datapoint. The heat map is given by the log predictive variance (as in Eq. 2.2.23) that encodes uncertainty in the latent space. The model is more likely to generate valid shapes from any location in the bright regions (*i.e.*, low variance regions).

Unlike Gaussian process based models, a standard DBN (or SBM) does not learn such a generative manifold. This implies, first of all, that a DBN does not allow us to sample "from the top" in a direct manner. Instead we must provide a test image to the visible units and condition on it before propagating it up and down the network for a few iterations to obtain an output sample. Secondly, like the VAE and InfoGAN, a DBN does not provide any predictive information about how plausible a generated sample is when we generate from a specific latent point.

In Figure 5.2 we provide an illustration of the learned manifold for the GPDBN model on the motorbike dataset.

We also performed a simple test by training a GPDBN on a single dataset containing two types of data (horses and motorbikes) to verify that the model learns a smooth manifold while at the same time maintaining a meaningful distinction between the different data by placing horses and motorbikes in two different regions of the manifold (Fig. 5.3).

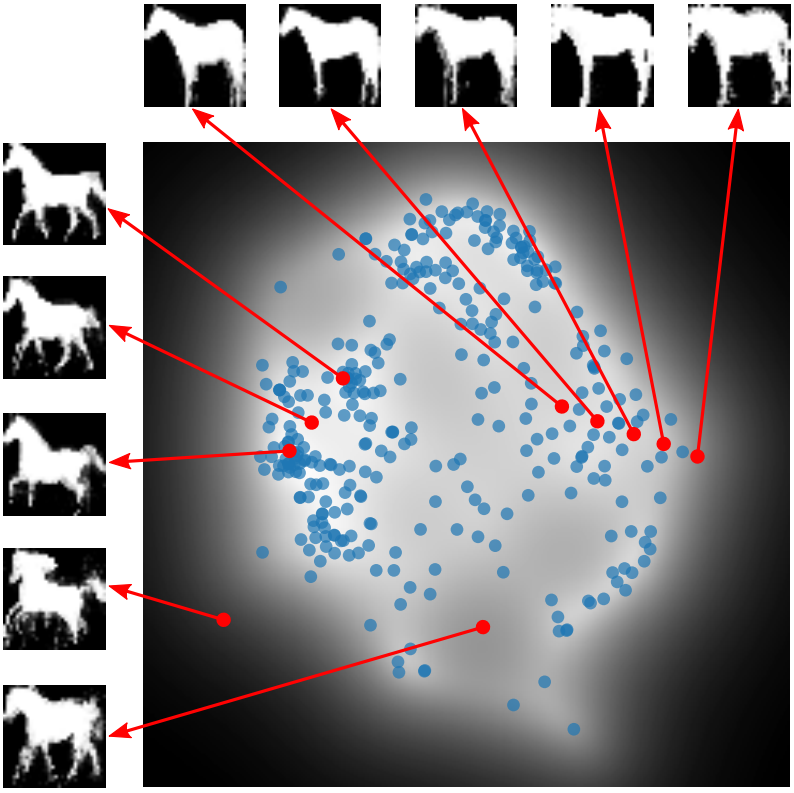


Figure 5.1: Example of manifold learned by the GPDBN model on the Weizmann horse dataset. Moving over the manifold changes the pose of the horse, with smooth paths in the manifold producing smooth transitions in silhouette pose. The heat map encodes the predictive variance of the model, with darker regions indicating higher uncertainty and lower confidence in the silhouette estimates.

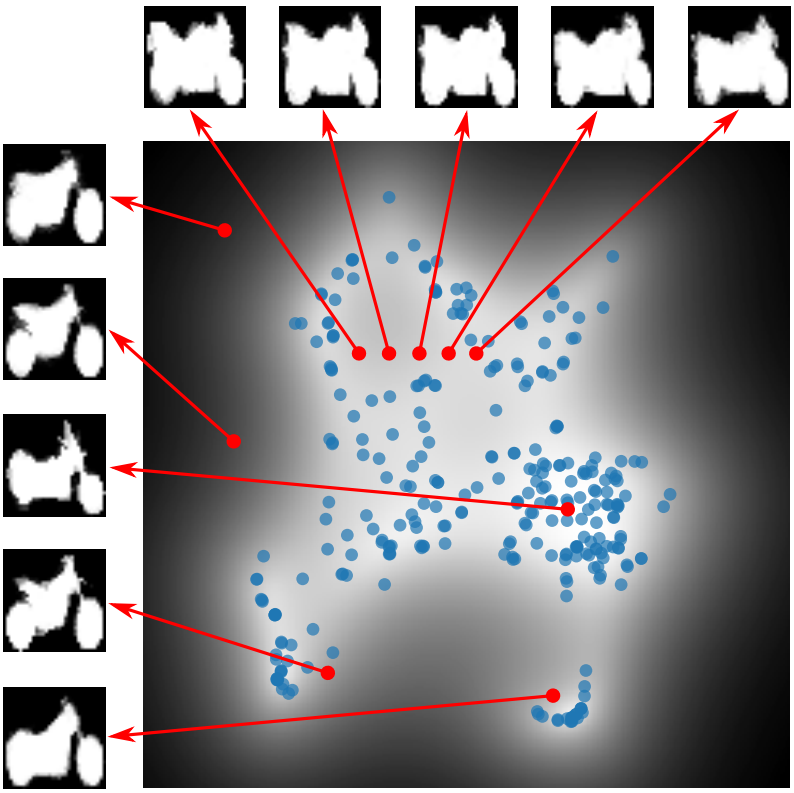


Figure 5.2: Manifold learned by the GPDBN model on the motorbikes dataset. Moving over the manifold changes the shape of the motorbike producing smooth silhouette transitions. The heat map encodes the predictive variance of the model, with darker regions indicating higher uncertainty and lower confidence in the silhouette estimates.

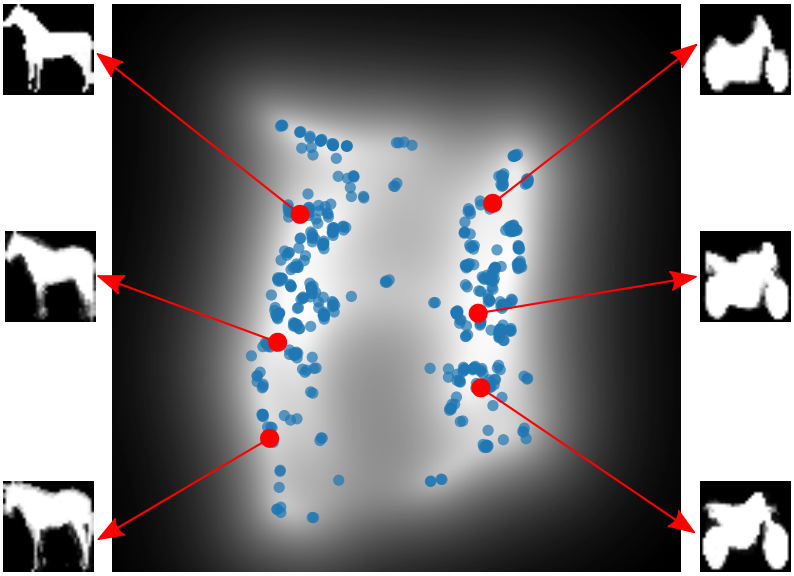


Figure 5.3: Manifold learned by the GPDBN model trained on 250 horses plus 250 motorbikes. Two different clusters are clearly visible while smoothness is still maintained.

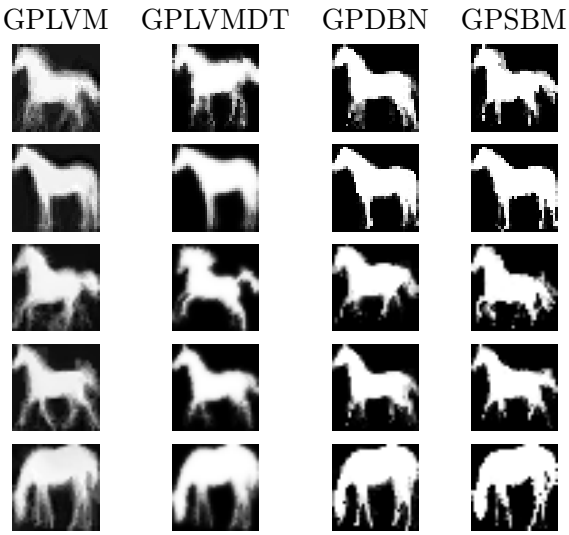


Figure 5.4: Qualitative comparison of silhouettes generated from low variance manifold areas by each of the models that provide uncertainty information in their learned latent spaces for easy synthesis (images manually ordered by visual similarity).

A smooth generative manifold, such the one learned by our model in Fig. 5.1 is informative as it gives us an indication about where to sample from to get plausible silhouettes. Fig. 5.4 compares silhouettes generated by the

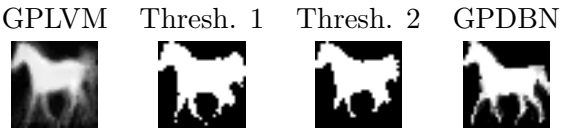


Figure 5.5: Comparison of a silhouette from the GPLVM, two thresholded versions of the same silhouette from the GPLVM, and one from the GPDBN.

models that allow sampling from the manifold.[1] We note that the GPLVM and GPLVMDT produce blurrier images since the shapes present interpolation artefacts from the Gaussian likelihood. In contrast, the results from both the GPDBN and GPSBM are sharper. Uncertainty in our models is concentrated in locations that are uncertain due to data quality, *e.g.* the thin legs and high variability of the tail.

Fig. 5.5 confirms the limitations of the direct GPLVM model where no threshold process can produce a reasonable silhouette. In contrast, the representation learned by the stochastic network of the GPDBN allows the model to predict sharper silhouettes.

5.5 Representation and Generalisation

In the recent literature on shape modelling, quantitative results are reported in terms of the distance between the test data not seen by the model and the most likely prediction under the model.

For the models that can be sampled from, this amounts to finding the location on the manifold that most closely represents the test input. We have discussed this for our models in Section 4.7. For the VAE, InfoGAN and ShapeOdds, for which we do not have access to predictive variance information in the latent space, the second term in Eq. 4.7.2 is ignored.

For a DBN (or SBM), the usual way to generate a sample is to condition on an observed sample and propagate it through the network for several cycles, as described in Section 2.1, with Gibbs samples taken after a burn in period. In our experiments, we fixed the conditioning on the test datapoint and averaged the results of a number of propagated samples through the model to prevent the sample chain from drifting away from the test data.

[1] When we show generated silhouettes from any model, we actually show grayscale images denoting pixel-wise probabilities of turning white rather than binary samples.

5.5.1 Projection under Noise

Objectively assessing unsupervised learning models is a fundamental problem in machine learning and there is no well-defined way to achieve this. Our proposal for a good quantitative assessment is to measure the reconstruction of the generated output corresponding to a noisy version of the input. To provide a challenging evaluation, we took unseen test data, corrupted it with noise and asked each model to find their most likely silhouette. Simply asking to "reconstruct" the test data (*i.e.*, finding closest silhouette to the original test datapoint without noise) would not be a sufficient evaluation since an identity mapping would be able to perform this task. In other words, a model that simply outputs the test datapoint (*i.e.*, an identity mapping) would achieve a perfect score in such a reconstruction test, however, under such a model all silhouettes would be equally likely (even implausible ones); generalising to implausible shapes is not a property that we want. Instead, we need the model to demonstrate that it can reject data (the noise) that should not be in the trained model. The model should be able to return a plausible reconstruction to the test datapoint as well as ignoring the noise. The predictive uncertainty of the GPDBN model gives us an indication of how plausible a generated silhouette is.

In Table 5.1, we report the results of our proposed models and the baseline methods for different percentages of salt-and-pepper noise in the test input (up to 60% to provide a more challenging environment where the test data has been corrupted by significant noise). We use the Structured Similarity measure (SSIM) (Wang et al., 2004) to perform quantitative evaluations since it is known to outperform both cross-entropy and mean squared error as a perceptual measure. The mathematical definition of the SSIM is the following:

$$\text{SSIM}(\boldsymbol{x}, \boldsymbol{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \ , \tag{5.5.1}$$

where, $\boldsymbol{x}$ and $\boldsymbol{y}$ are the two input images with M number of pixels,

$$\mu_x = \frac{1}{M}\sum_{i=1}^{M} x_i \ , \tag{5.5.2}$$

$$\sigma_x^2 = \frac{1}{M}\sum_{i=1}^{M}(x_i - \mu_x)^2 \ , \tag{5.5.3}$$

$$\sigma_{xy} = \frac{1}{M}\sum_{i=1}^{M}(x_i - \mu_x)(y_i - \mu_y) \ , \tag{5.5.4}$$

Method	10% noise	20% noise	60% noise
ShapeOdds	0.49 ± 0.06	0.43 ± 0.06	0.25 ± 0.04
DBN	0.47 ± 0.10	0.43 ± 0.10	0.27 ± 0.05
SBM	0.55 ± 0.10	0.54 ± 0.11	0.35 ± 0.05
VAE	0.42 ± 0.09	0.36 ± 0.08	0.11 ± 0.02
InfoGAN	0.33 ± 0.10	0.27 ± 0.06	0.18 ± 0.03
GPLVM	0.44 ± 0.07	0.48 ± 0.07	0.44 ± 0.07
GPLVMDT	0.54 ± 0.09	0.54 ± 0.09	0.37 ± 0.03
GPDBN	0.56 ± 0.10	0.54 ± 0.12	0.42 ± 0.10
GPSBM	0.58 ± 0.09	0.59 ± 0.08	0.51 ± 0.09

Table 5.1: Mean and standard deviation of the SSIM score between horse silhouettes from each model against the original test data without noise for the task of finding a good reconstruction of the data which was corrupted with different noise levels (10%, 20% and 60%).

C_1 and C_2 are two small arbitrary positive constants to avoid instability. The output of the SSIM ranges from 0 to 1 (higher values are better, with 1 meaning that the inputs are identical). The actual implementation of the SSIM that we used is the one provided by the Scikit-image Python library (Scikit-image, 2019) (we used default values for all the parameters and set "win_size" to 3, this defines side-length of the sliding window used in the comparisons).

A random sample of corresponding silhouettes for the horse dataset is provided in Figures 5.6, 5.7 and 5.8.

In Table 5.2 we provide results for the motorbike dataset. Figures 5.9, 5.10 and 5.11 show examples of corresponding output from the models.

The quantitative comparisons show that our GPDBN and GPSBM models have captured a high quality probabilistic estimate of the data manifold while also providing the interpretable manifold and explicit representation of model uncertainty.

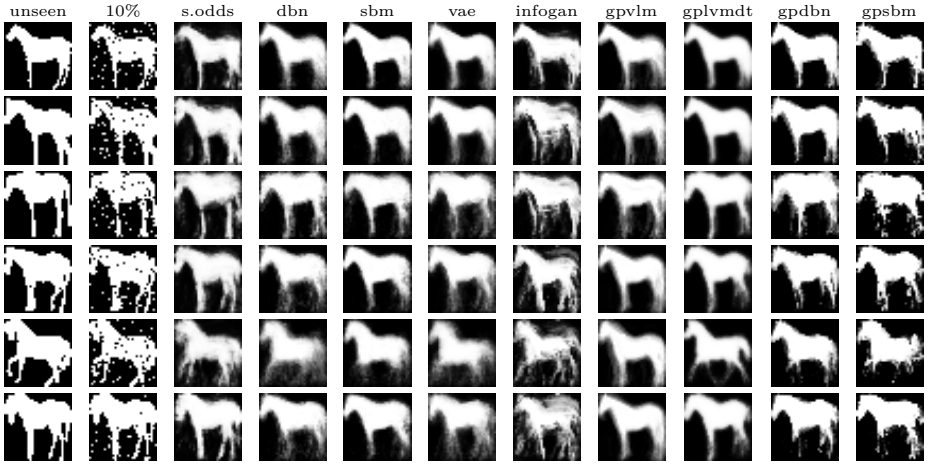


Figure 5.6: Test silhouettes (first column) are corrupted with 10% salt and pepper noise (second column). The remaining columns show estimated silhouettes from each model.

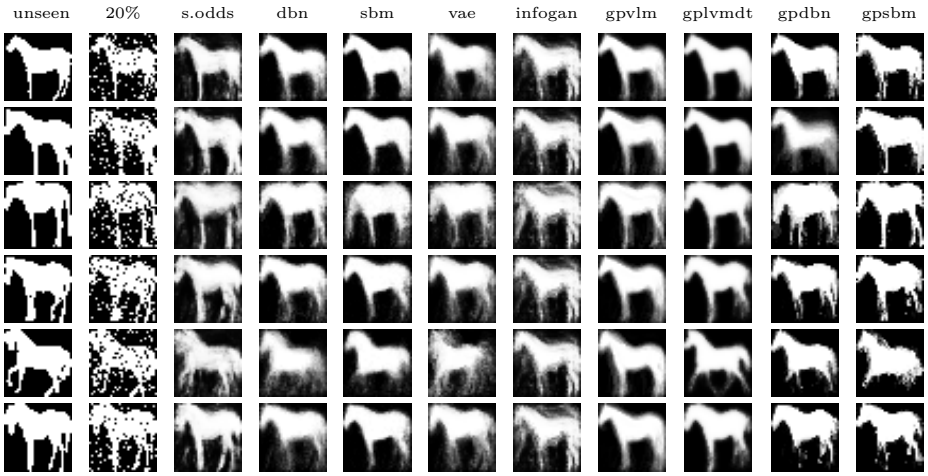


Figure 5.7: Test silhouettes (first column) are corrupted with 20% salt and pepper noise (second column). The remaining columns show estimated silhouettes from each model.

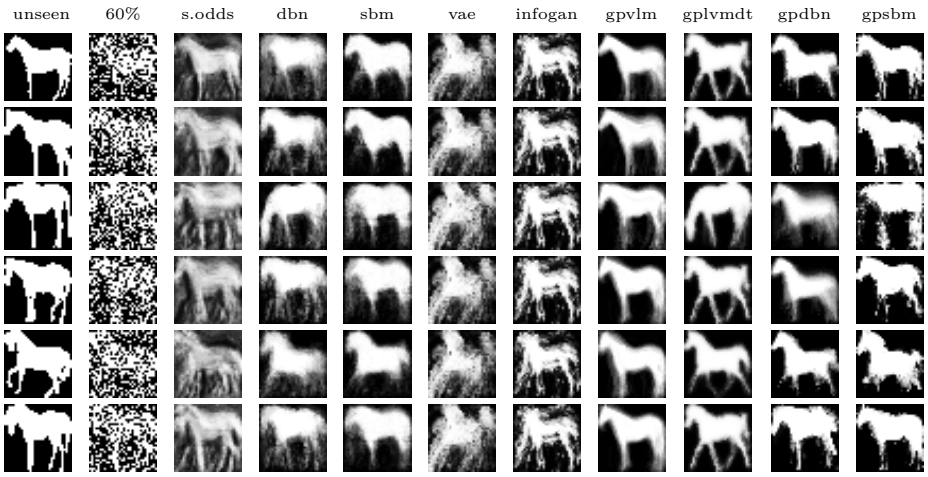


Figure 5.8: Test silhouettes (first column) are corrupted with 60% salt and pepper noise (second column). The remaining columns show estimated silhouettes from each model.

Method	20% noise	40% noise	60% noise
ShapeOdds	0.53 ± 0.09	0.41 ± 0.08	0.28 ± 0.05
DBN	0.42 ± 0.03	0.42 ± 0.03	0.42 ± 0.04
SBM	0.58 ± 0.04	0.57 ± 0.04	0.55 ± 0.05
VAE	0.35 ± 0.11	0.17 ± 0.02	0.17 ± 0.03
InfoGAN	0.54 ± 0.05	0.45 ± 0.04	0.50 ± 0.04
GPLVM	0.57 ± 0.04	0.55 ± 0.04	0.51 ± 0.06
GPLVMDT	0.58 ± 0.04	0.55 ± 0.04	0.54 ± 0.05
GPDBN	0.64 ± 0.03	0.58 ± 0.05	0.55 ± 0.07
GPSBM	0.63 ± 0.02	0.61 ± 0.03	0.55 ± 0.07

Table 5.2: Mean and standard deviation of the SSIM score between motorbike silhouettes from each model against the original test data without noise for the task of finding a good reconstruction of the data which was corrupted with different noise levels (20%, 40% and 60%).

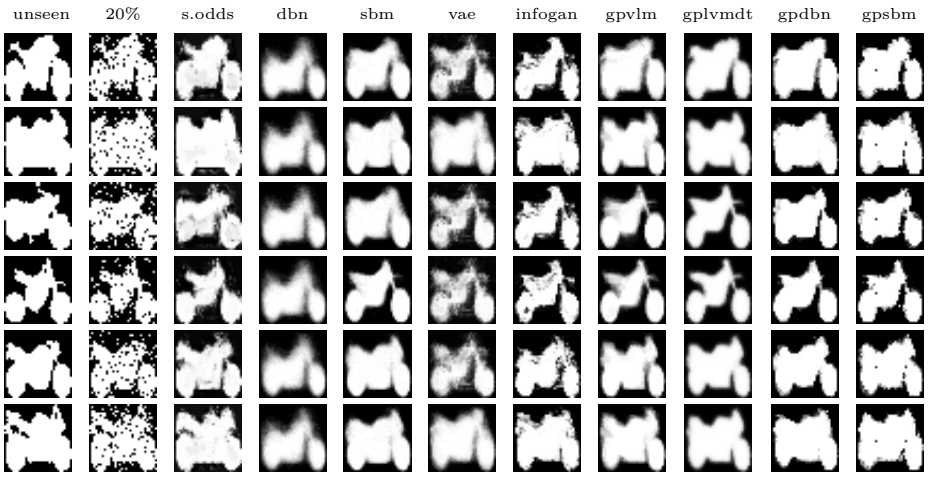


Figure 5.9: Test silhouettes (first column) are corrupted with 20% salt and pepper noise (second column). The remaining columns show estimated silhouettes from each model.

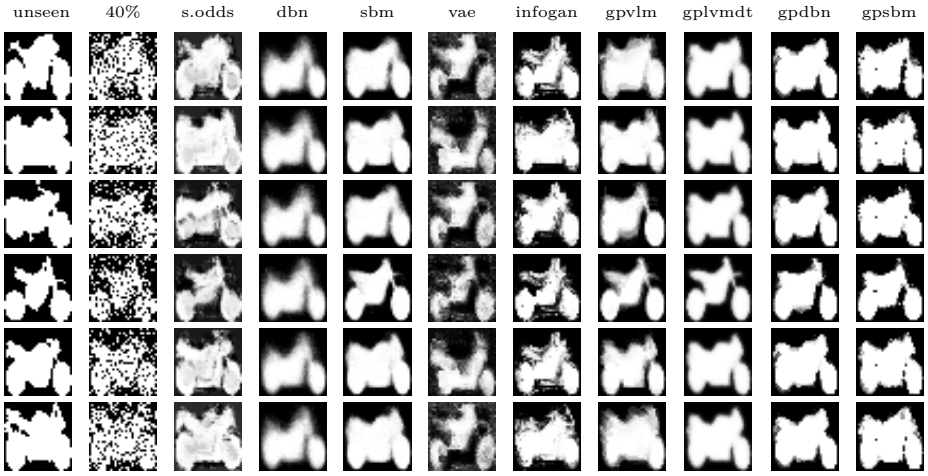


Figure 5.10: Test silhouettes (first column) are corrupted with 40% salt and pepper noise (second column). The remaining columns show estimated silhouettes from each model.

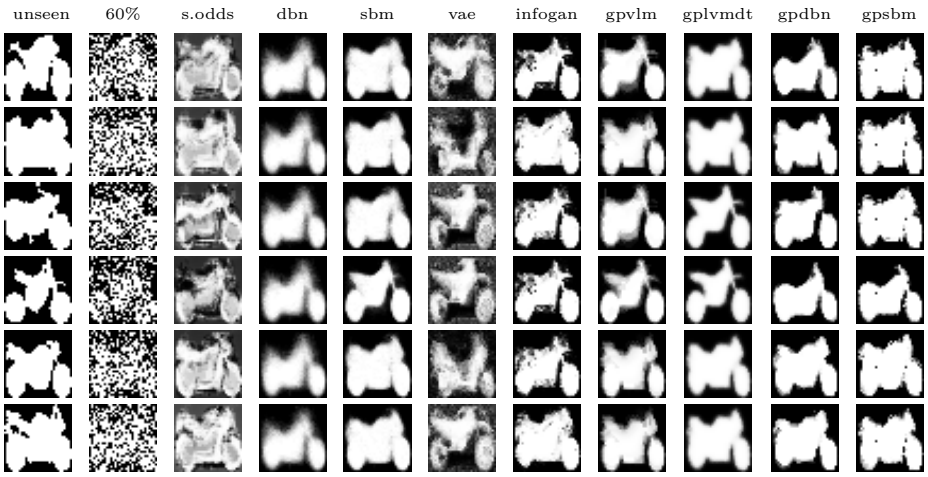


Figure 5.11: Test silhouettes (first column) are corrupted with 60% salt and pepper noise (second column). The remaining columns show estimated silhouettes from each model.

5.6 Smoothness: Interpolation Test

We trained a GPDBN, VAE and InfoGAN models on a 30-image dataset (which we call *stars* dataset) generated from a *known* 1-dimensional manifold using a simple script. The full dataset is displayed in the top row of Fig. 5.12. The deterministically generated dataset allows us to determine quantitatively whether interpolations in the latent space are representative of the true data distribution.

The bottom rows in Fig. 5.12 show the model outputs for the interpolation between two latent points corresponding to a four-pointed *star* (leftmost sample) and a *square* (rightmost sample). The uncertainty information of the GPDBN allows us to go from one point to the other passing through low-variance regions by following a geodesic path (Tosi et al., 2014). We can see that the GPDBN produces smoothly varying shapes of high quality that reflect the true manifold. In contrast, the VAE and InfoGAN results do not smoothly follow the true manifold and contain some erroneous interpolants that do not reflect the true data; this is supported by the quantitative results that measure the quality of the samples to the true data using SSIM (Table 5.3).

The ability to exploit variance information in the GPDBN is clearly an advantage over the VAE and InfoGAN where the absence of direct access to the latent predictive posterior distribution prevents easy access to geodesics.

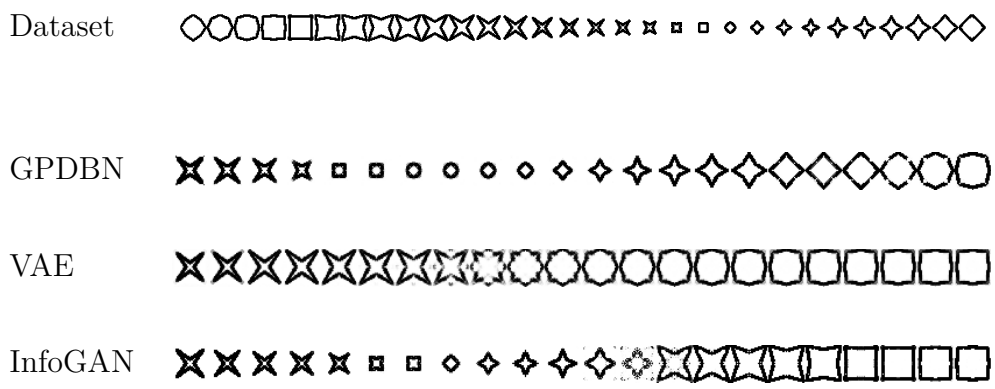


Figure 5.12: Example results of the interpolation test between two training points from the stars dataset. The top row shows the geodesic interpolation generated by the GPDBN. The following rows are the linear interpolations generated by the VAE and InfoGAN respectively. (In this particular picture black and white are inverted respect to the data.)

Method	SSIM
GPDBN	0.95 ± 0.01
VAE	0.87 ± 0.03
InfoGAN	0.93 ± 0.06

Table 5.3: Mean and standard deviation of the SSIM score over 10 interpolation experiments for each model. A higher score is better.

5.7 Scaling Experiments

In Fig. 5.13 we compare the performance of the GPDBN, InfoGAN and VAE models as the size of the training dataset increases; here we used the MNIST digit dataset. We used a 10-dimensional latent space for all of the three models to account for the larger quantity of data. In a similar way to the experiments described in Section 5.5.1, we took 30 random images from the MNIST test data, added 20% salt-and-pepper noise and then we calculated the SSIM score between the output of the models and the test data without noise. We plotted the score against dataset size (in log scale).

In Fig. 5.13 we can observe that the GPDBN model is able to capture a high quality model of the data manifold even from small datasets; for example, it achieves the same quality as a VAE trained on 10000 images using only 100. We argue that the propagation of uncertainty throughout the model provides the advantage over both the VAE and InfoGAN, which are both trained with

only maximum likelihood approaches.

In addition to the scalability experiment in Fig. 5.13 we provide in Figures 5.14 and 5.15 two examples of learned latent spaces to show that our approach overcomes scaling issues normally present in Gaussian process models and deep belief networks. Fig. 5.14 is an example of the latent space of a GPDBN trained on the 60000 MNIST images via our proposed mini-batching approach (Sec. 4.6).

In Fig. 5.15 we show an example of learned latent space for a GPDBN trained on higher resolution images from the horse dataset (128×128). By using convolutional architectures, we can scale the number of parameters similarly to convolutional feed-forward networks, and the fully stochastic network allows us to train from random weight initialisation using backpropagation without the need to use contrastive divergence.

However, we note that, in comparison to the output of to the standard GPDBN architecture, the added convolutions seem to add some artefacts. This is likely due to the fact that the number of training images in the dataset is the same although their size is much larger, also the higher number of network layers add some extra complexity to the model.

With both these approaches (*i.e.*, the mini-batching training model and convolutional network model) we still maintain full uncertainty propagation throughout the model.

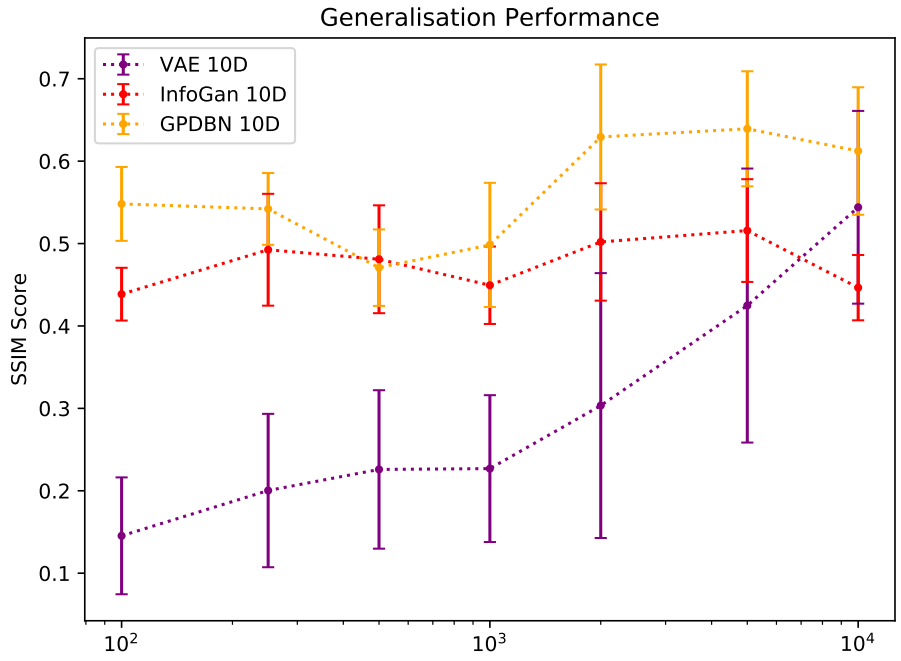


Figure 5.13: Graph showing the SSIM score of the output of the GPDBN, InfoGAN and VAE models against the test data without noise as the training dataset size increases from 100 to 10000 datapoints. A higher score is better.

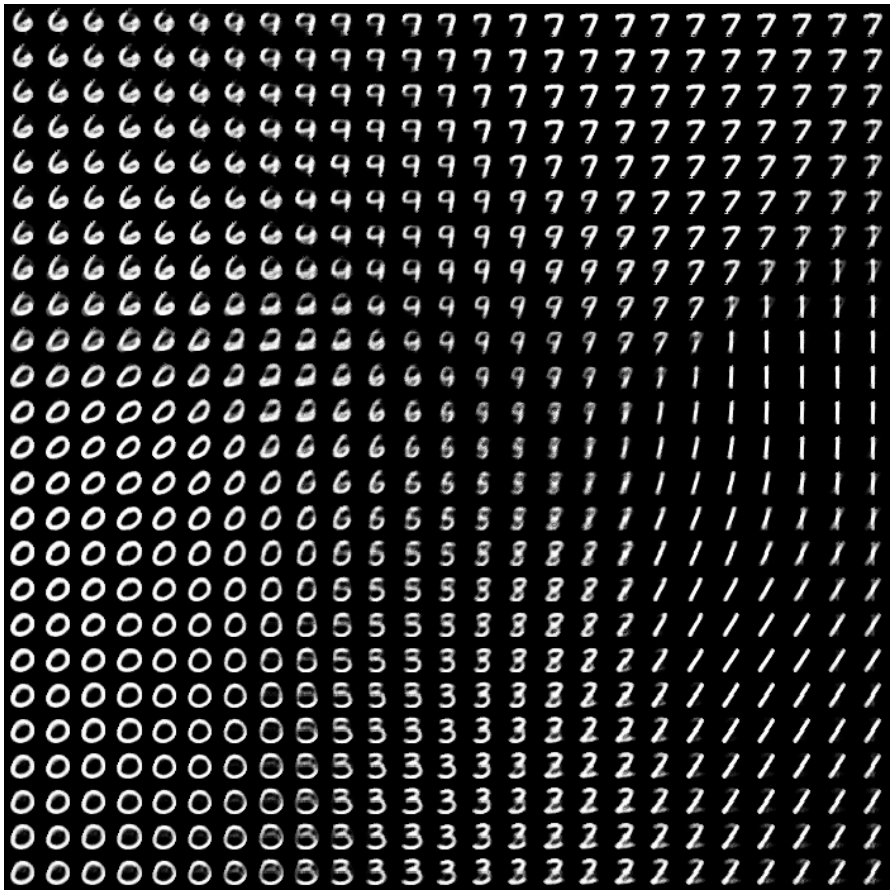


Figure 5.14: The model can be made to scale to a large number of datapoints by optimising the objective using mini-batching (Sec. 4.6). This picture was produced by a GPDBN trained using mini-batching (with mini-batch size of 250) on MNIST comprising of 60000 28×28 images.

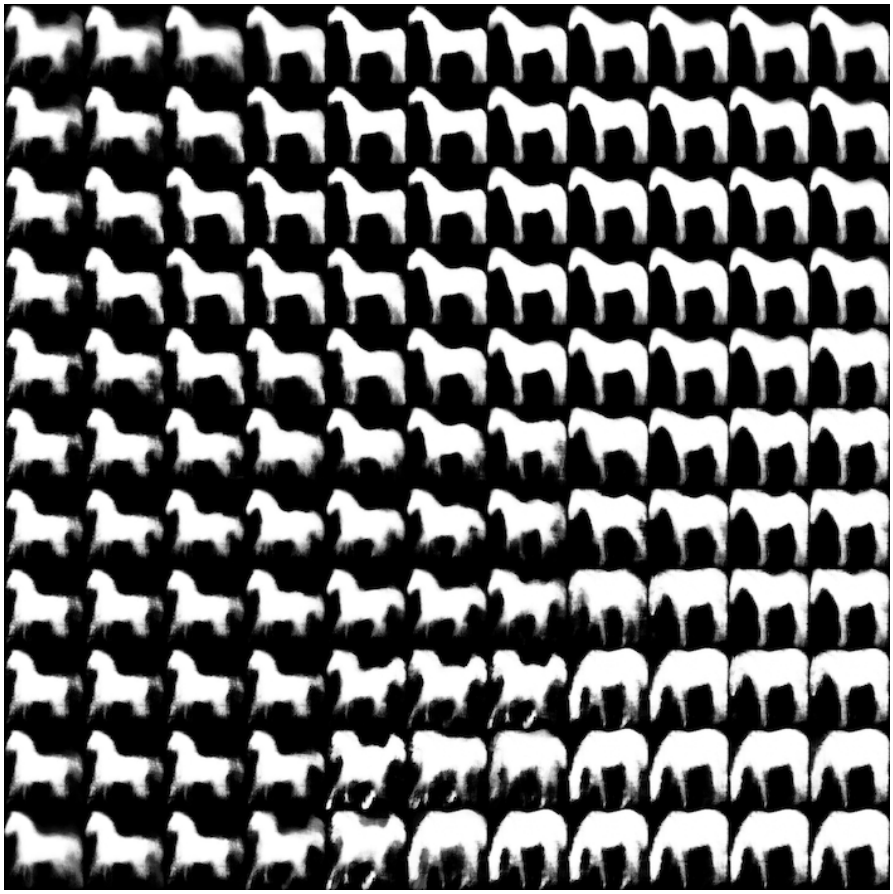


Figure 5.15: Scalability in the size of images can be obtained by using convolutions in the lower layers. This picture was obtained by training a GPDBN on the Weizmann Horses comprising of 328 128×128 images. Here the stochastic network has 4 layers in total, of which, the two lowest layers are convolutional and use a filter size of 7 and 15 respectively.

5.8 Smoothness Experiment

This additional experiment further highlights the benefit of the uncertainty associated with the proposed model and how it manifests itself in the proposed models in contrast with the other methods. There is an inherent trade-off between a simple topology of the manifold and the smoothness of the mapping. To exemplify this we generated a dataset of a simple shape that is deformed in a cyclic manner, we generated 30 silhouettes from a smooth 1D manifold (*i.e.*, a circle) that parameterises the joints of this simple shape (top row in Fig. 5.16). Ideally, we would like the model to recover this smooth manifold by encoding the training data on a continuous 1D trajectory that interpolates the training examples smoothly and correctly. Importantly, if the uncertainty in the model reflects that of the data, we should move along ridges of high probability (manifold geodesics) to generate realistic data. Our proposed model is directly applicable to such approaches as described in the work of Tosi et al. (2014). Further, the experiment highlights how the uncertainty information reflects the prediction. When generating shapes corresponding to a region of the manifold where the model is highly uncertain we would, if the model has captured the characteristics of the data well, expect images corresponding to the *average* shape. As can be seen, the GPDBN and GPSBM generate an average shape while the other methods fail to capture this characteristic in the data, making it challenging to interpret the uncertainty.

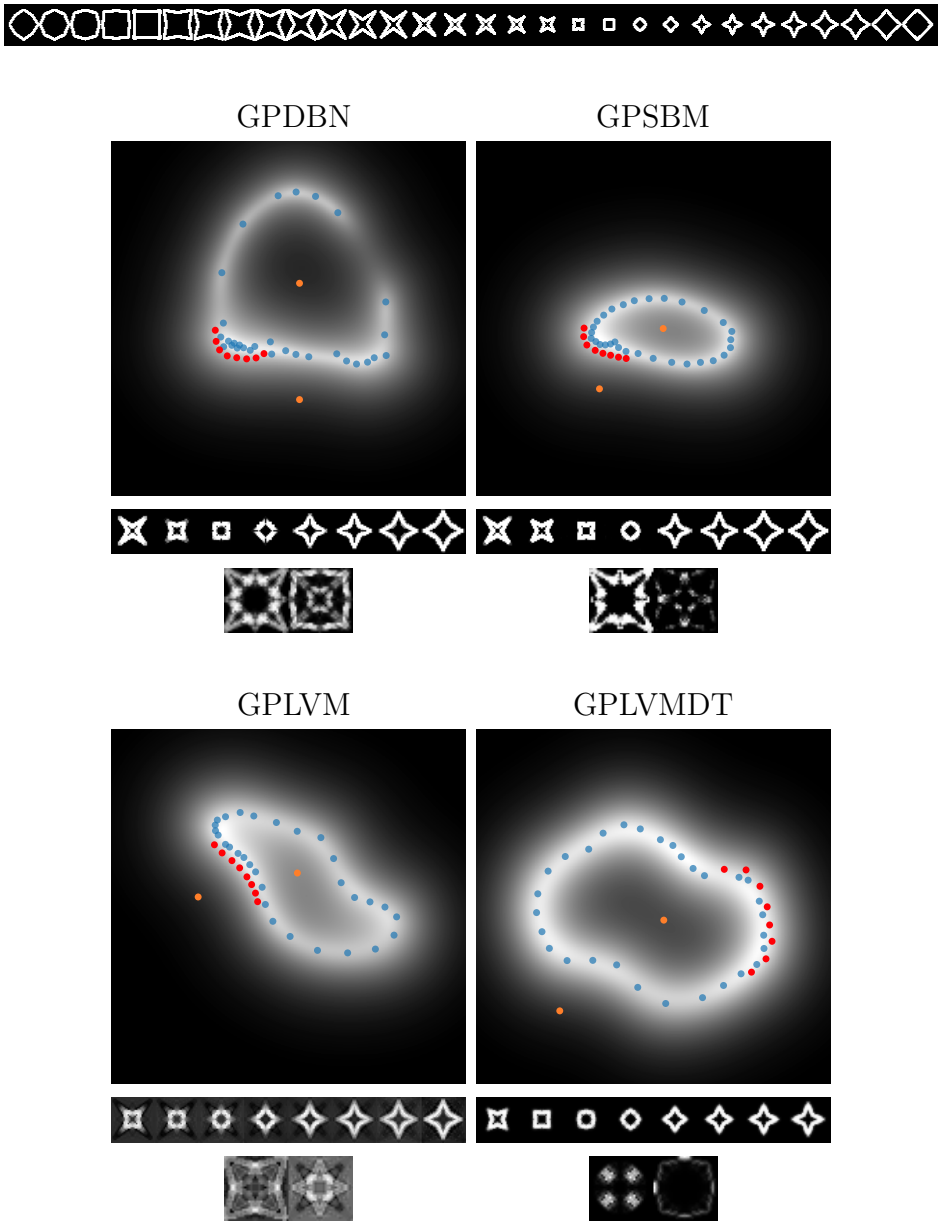


Figure 5.16: Comparison of manifold models trained on the star dataset consisting of 30 points on a 1D manifold mapped to a corresponding set of silhouettes (top row). The blue points on each manifold are the latent locations corresponding to the training examples from the top row. The 8 red points are test locations on a smooth path on the manifold; the corresponding 8 novel silhouettes generated at these points are shown below each manifold. The lowest row below each manifold shows 2 silhouettes corresponding to the 2 orange points away from the low-variance manifold.

Structure

The blue points on the manifolds corresponding to the training data in Fig. 5.16 show the topology of the manifolds learned by each of the four smooth manifold-based models. We note that, in this experiment, a few iterations of pre-training for the networks of the GPDBN and GPSBM was used to avoid unnecessary "folds" in the learned manifolds due to possible bad network weights initialisations, allowing a fairer comparison. All the manifolds have correctly identified a smooth trajectory for the training data as well as capturing the periodic repetition by closing the path.

The 8 red points on each manifold represent test locations corresponding to the 8 samples below each manifold in Fig. 5.16. Here we see that all models are correctly interpolating the overall pattern, however, the Gaussian likelihood of the GPLVM introduces artefacts in the silhouettes that are not found in the results from the GPDBN and GPSBM. The GPLVMDT improves over the GPLVM but still produces blurred results which do not reflect the sharp contours of the shapes in the original data.

Uncertainty

The real power of the GPDBN model is captured by looking at what happens when we leave the manifold. The final row of silhouettes are samples from the orange points that are in regions of high predictive variance (low confidence). Whereas the GPDBN and GPSBM return shapes that reflect well the uncertainty in the manifold (the GPDBN and GPSBM results are the mean of a set of samples from the model and away from the manifold these results are correctly approaching the mean of the training data); both the GPLVM and the GPLVMDT produce unreasonable results outside the manifolds, with the GPLVM returning a pixel-average that is strongly affected by the artefacts, and the GPLVMDT returning implausible shapes.

5.9 Ablation and Fine-tuning

In this chapter we have provided extensive results and comparisons with recent models. Here we provide a simple study of feature ablation and model fine-tuning.

Ablation of the Main Model Components

We note that the two components of the GPDBN model, that is the GPLVM and the stochastic network (our DBN), are both essential, none of them can be ablated since the former provides the smooth manifold and predictive uncertainty while the latter increases the capability of generating good image data. Moreover, the comparisons provided in this chapter show that both GPLVM and DBN are weaker as standalone models.

Ablation of Concrete Units

Dropout is a popular way to introduce stochasticity into a neural network (Srivastava et al., 2014). Differently from a Concrete unit, a dropout unit is a normal unit (*e.g.*, sigmoidal) whose output is dropped (*i.e.*, set to 0) with probability p_d (*i.e.*, a fixed parameter, usually 0.1). In Fig. 5.17 we show that ablation of our fully stochastic network with Concrete units to dropout units degrades uncertainty propagation and reduces the performance of the model. As for the experiments in Section 5.5.1, we used 30 random images from the MNIST test data, corrupted them with 20% salt-and-pepper noise, and we then calculated the SSIM score between the output of the models and the test data without noise. We plotted the score against the training dataset size in log scale.

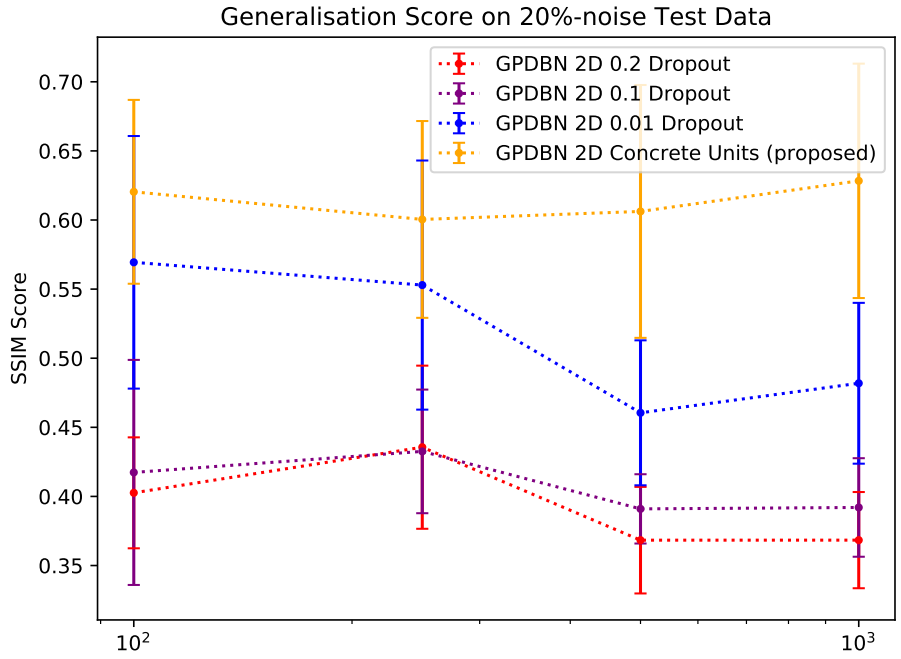


Figure 5.17: Ablation comparisons of GPDBN model variants using sigmoidal units with dropout instead of Concrete units. In terms of number of units and layers the architecture of the compared models matches exactly the standard GPDBN. Blue, purple and red curves are model variants that use sigmoidal units with dropout probabilities of 0.01, 0.1 and 0.2 respectively. This graph shows the importance of Concrete units for proper uncertainty propagation.

Varying the Network Architecture

The architecture of the GPDBN that we used in the experiments in Chapter 5 consists of a stochastic network with three layers of 200 Concrete units, 100 Concrete units and 50 Gaussian units (from bottom to top). Admittedly, this is not necessarily the best network architecture; we could have chosen a different number of layers and units. Finding the right number of units and layers is an open problem in deep learning (Goodfellow, Bengio and Courville, 2016, p. 198). The *universal approximation theorem* (Cybenko, 1989; Hornik, Stinchcombe and White, 1989) states that a simple neural network with a single hidden layer and any *squashing* activation function (such as the sigmoid) can represent a wide variety of functions. However, in order for this to be valid a very large number of units is required in the hidden layer. Bengio and LeCun (2007) suggest that deeper architectures can represent complex functions more efficiently. Based on these considerations and given the small sizes of the

datasets in our experiments we used a three-layer network architecture, this is also to allow fair comparisons with the other models, such as DBN and VAE (for which these considerations are also valid), for which we used the same number of layers and units to our model (as described in Sec. 5.2). Furthermore, for simplicity, we kept the network architecture in our model fixed for all our experiments.

We trained a GPDBN on the horse dataset experimenting with four different stochastic networks (increasing and decreasing the number of units and layers). As shown in Table 5.4, tripling the number of units at all layers or reducing the number of units by 1/3 or adding an additional layer did not lead to performance gains compared to the corresponding results for our generic GPDBN with a three-layer network architecture (that we proposed for our experiments reported in Table 5.1 with 20% noisy data). Instead, in the case of removing one layer we obtained slightly better performance (Tab. 5.4). We interpret this finding positively as it suggests that the model can be further fine-tuned to achieve even higher performance.

Finding the optimal number of layers and weights is a common issue with many deep learning methods. We think that using a Bayesian approach with sparsity inducing priors to prune parts of the network to use the optimal architecture automatically would be the right way to tackle this problem. This needs further investigation and could be a good line for future work.

Method	SSIM
Large net (x3 units)	0.52 ± 0.12
Narrow net (1/3 units)	0.47 ± 0.15
Deep net (+1 layer)	0.45 ± 0.20
Shallow net (-1 layer)	0.58 ± 0.08

Table 5.4: SSIM of the output against test data (without 20% salt-and-pepper noise) of the GPDBN using different network architectures.

Chapter 6

Dropout Architectures

In Section 5.9 we have shown that the ablation of the Concrete units in the GPDBN in favour of simple sigmoidal units with dropout reduces the performance of the model. It is possible to extend the dropout idea and introduce stochasticity in the network to propagate probabilistic uncertainty in more elaborate ways. In this section we briefly introduce some alternative dropout ideas and provide additional experimental results in comparison with the standard GPDBN model.

6.1 Standard Dropout

The standard dropout method (Srivastava et al., 2014) uses a single parameter, that we denote with q, to specify the probability of dropping the unit's output (*i.e.*, setting it to 0). The q parameter is fixed to the same value for all the units in the layer. More formally, this can be formulated as follows:

$$\boldsymbol{h}_2 = \text{Sigmoid}(\boldsymbol{W}^\top \boldsymbol{h}_1 + \boldsymbol{b}) \odot \boldsymbol{m} \ , \tag{6.1.1}$$

here, $\boldsymbol{h}_2$ denotes the output of the layer, $\boldsymbol{W}$ is the layer's matrix of weight parameters, $\boldsymbol{b}$ is a vector of bias parameters and $\boldsymbol{h}_1$ is the input vector of the activations from the previous layer. The most important component in Eq. 6.1.1 is the binary mask $\boldsymbol{m}$ which multiplies the sigmoid element-wise. Denoting with u_i a sample from a uniform distribution $\mathcal{U}(0, 1)$, a single value

m_i (corresponding to the *i-th* unit) in the mask vector $\boldsymbol{m}$ is obtained as follows:

$$m_i = \begin{cases} 0 & \text{for } u_i \leq q \ , \\ 1 & \text{for } u_i > q \ . \end{cases} \tag{6.1.2}$$

6.2 Binary Units

A stochastic binary layer of a deep belief network can be formulated in terms of dropout. The layer's output is simply the binary mask:

$$\boldsymbol{h}_2 = \boldsymbol{m} \ , \tag{6.2.1}$$

each value m_i in the output vector $\boldsymbol{m}$ is computed as in Eq. 6.1.2, except for the fact that here q is not a fixed parameter but it is instead a parametrised expression that directly depends on the parameters of each single unit:

$$q_i = 1 - \text{Sigmoid}(\boldsymbol{w}_i^\top \boldsymbol{h}_1 + b_i) \ , \tag{6.2.2}$$

here, $\text{Sigmoid}(\boldsymbol{w}_i^\top \boldsymbol{h}_1 + b_i)$ is the probability of retaining the *i-th* unit's output. It is also important to note that while in a standard dropout network we can differentiate with respect to the layer parameters (*i.e.*, $\boldsymbol{W}$ and $\boldsymbol{b}$), here instead, the binary sampling using the step function of Eq. 6.1.2 prevents us from differentiating with respect to the parameters.

6.3 Concrete Units

The idea behind the Concrete units layer which we use in the GPDBN model is a direct extension of the formulation of the binary stochastic layer in terms of dropout that we have provided in Sec. 6.2. We would like to be able to differentiate with respect to the parameters through the sampling process; it is possible to achieve this by replacing the dropout step function of Eq. 6.1.2 with a differentiable function, a continuous relaxation such as the Concrete function (Maddison, Mnih and Teh, 2017):

$$\text{Con}\,(p_i, u_i) = \text{Sigmoid}\left[\frac{1}{\lambda}\big(\log p_i - \log(1-p_i) + \log u_i - \log(1-u_i)\big)\right] \ , \tag{6.3.1}$$

here, p_i represents the Bernoulli probability of retaining the unit's output (which is equivalent to $1 - q_i$), λ is a scaling factor (normally, 0.1) which

controls the steepness of the function (Fig. 4.2), u_i is a sample from the uniform distribution. Importantly, since in Eq. 6.3.1 the sample u_i does not depend on p_i, the functional relation between u_i and the output sample is differentiable with respect to p_i. Specifically, the function is differentiable with respect to the unit's parameters as we define p_i as follows:

$$p_i = \text{Sigmoid}(\boldsymbol{w}_i^\top \boldsymbol{h}_1 + b_i) \ . \tag{6.3.2}$$

Similarly to the binary layer (Sec. 6.2), the output of the Concrete layer is simply the vector $\boldsymbol{m}$:

$$\boldsymbol{h}_2 = \boldsymbol{m} \ , \tag{6.3.3}$$

where each single value m_i is sampled from the Concrete distribution:

$$m_i \sim \text{Con}\,(p_i, u_i) \ . \tag{6.3.4}$$

6.4 Concrete Dropout

The main idea in Concrete Dropout introduced by Gal, Hron and Kendall (2017) is the substitution of the dropout binary step function (6.1.2) with the Concrete function (6.3.1). Similarly to standard dropout (Sec. 6.1), the layer's output is defined as follows:

$$\boldsymbol{h}_2 = \text{Sigmoid}(\boldsymbol{W}^\top \boldsymbol{h}_1 + \boldsymbol{b}) \odot \boldsymbol{m}/p \ . \tag{6.4.1}$$

Each value m_i of the mask vector $\boldsymbol{m}$ is sampled from the Concrete distribution:

$$m_i \sim \text{Con}\,(p, u_i) \ , \tag{6.4.2}$$

where p is a learnable parameter which is shared among all the units and represents the probability of retaining the output of each unit. We also note that in Eq. (6.4.1), dividing by p prevents variations in the expected sum of the output of the units.

6.5 Parametrised Concrete Dropout

The idea of Concrete Dropout (Sec. 6.4) can be extended to replace the single shared parameter p with an expression that directly depends on the unit's parameters; we call this method *parametrised Concrete dropout*. Thus, we

define p_i for each units as follows (as for the Concrete Units in Sec. 6.3):

$$p_i = \text{Sigmoid}(\boldsymbol{w}_i^\top \boldsymbol{h}_1 + b_i) \ . \tag{6.5.1}$$

As in Concrete dropout, the layer's output is defined as follows:

$$\boldsymbol{h}_2 = \text{Sigmoid}(\boldsymbol{W}^\top \boldsymbol{h}_1 + \boldsymbol{b}) \odot \boldsymbol{m}/\boldsymbol{p} \ , \tag{6.5.2}$$

here, $\boldsymbol{p}$ is the vector of all the single parametrised expressions p_i, and $\boldsymbol{m}$ is the mask vector where each m_i is sampled as follows:

$$m_i \sim \text{Con}\,(p_i, u_i) \ . \tag{6.5.3}$$

Our hypothesis to use an expression that depends on the unit's parameters rather than a shared p is to allow more flexibility for the single units while at the same time avoiding to introduce new parameters in the model. Our experimental results (Sec. 6.6), however, do not reflect any significant performance gain compared to Concrete dropout.

6.6 Experimental Results

In Figure 6.1 we provide a comparison of the performance of the GPDBN baseline (with Concrete units) and three variants of the GPDBN model where the DBN is replaced with each of the dropout architectures described in the previous sections: Standard Dropout (Sec. 6.1), Concrete dropout (Sec. 6.4) and Parametrised Concrete dropout (Sec. 6.5). In terms of number of layers and units, the network structure of all the compared models is exactly the same. The experiment setup matches the generalisation experiments in Chapter 5: 30 images are randomly chosen from the MNIST test data, they are then corrupted with 20% salt-and-pepper noise, the SSIM score is calculated between the output of the models and the test data without noise (the training dataset size in the graph is reported in log scale). Each point in the graph is calculated as the average of 10 experiment runs (*i.e.*, for each dataset size each model is trained and tested 10 times, this is to avoid misleading results that may arise due to the high stochasticity of the models).

The downward trend of the dropout architectures in the graph in Figure 6.1 suggests that as the dataset size increases the dropout models tend to generalise less compared to the standard GPDBN with concrete units. Furthermore, we

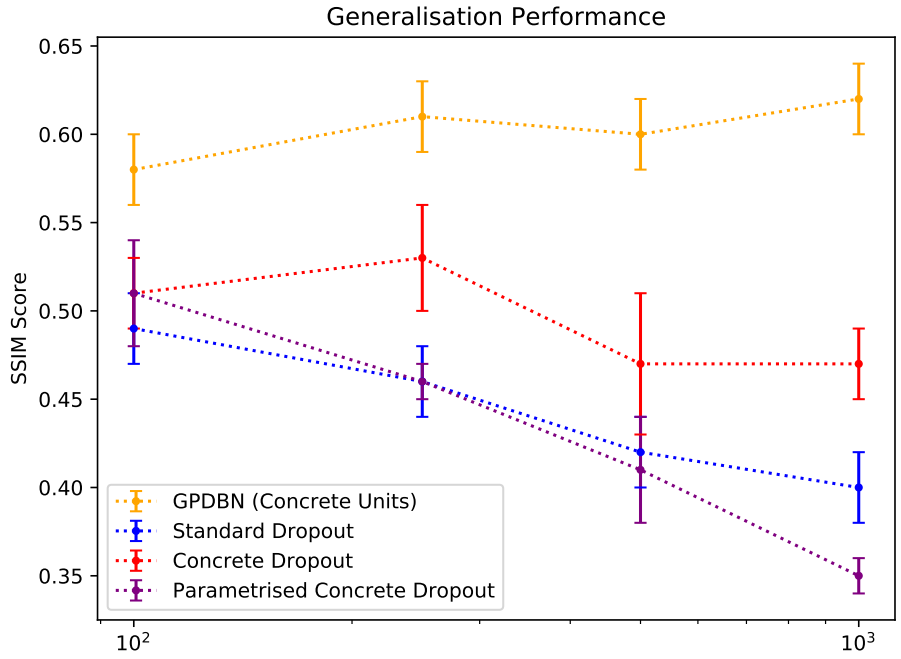


Figure 6.1: Graph showing the SSIM score of the output of the baseline GPDBN with Concrete units, standard sigmoidal dropout units, Concrete dropout units and parametrised Concrete dropout units against the test data without noise as the training dataset size increases from 100 to 10000 datapoints. A higher score is better.

note that there is a larger gap between the performance of the GPDBN compared to the other dropout model variants, suggesting that the use of Concrete units is the optimal way to introduce stochasticity in the network and propagate uncertainty. This downward trend for the generalisation score as the training data increases is also confirmed in Fig. 6.2. We compared the standard dropout model with two variants of the same model: one with a narrower network structure (with half the number of units at each layer), and another model with a larger network (double the number of units at all layers). The graph shows that as the data increases the difference in the performances of the models reduces, suggesting that the difference in the network structure is not as important as the way uncertainty is propagated.

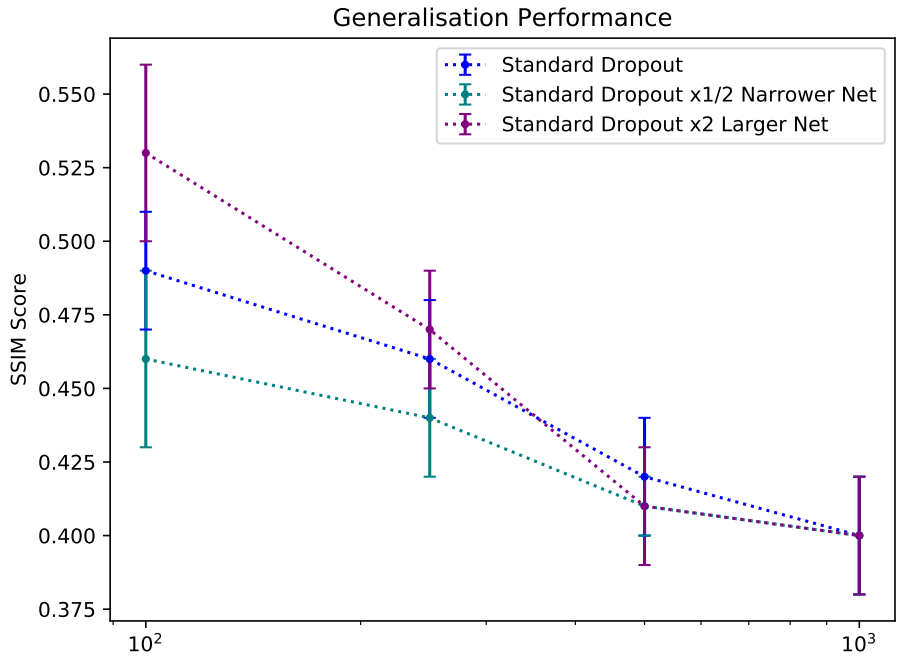


Figure 6.2: Graph showing the SSIM score of the output of standard dropout models with sigmoidal units against the test data without noise as the training dataset size increases from 100 to 10000 datapoints. The blue curve is the standard dropout model with 200, 100 and 50 units (from bottom to top); the green curve is the same model with half the number of units at all layers; the purple curve is the model with double the units at all layers.

Chapter 7

Remarks on Generalisation

A common definition of *generalisation* is "the formulation of general concepts from specific instances by abstracting common properties" so that a conceptual model can be created (Wikipedia, 2019).

For example, within the context of representation learning, in classification, a related statement is that a model generalises well when the gap between the performance on the training set and on the test set is small, implying that the model has created good abstractions from the instances on which it is trained. These definitions, however, are very generic and might lead to misleading conclusions if we are not careful with our assumptions.

To illustrate this, let us consider a simple example. We train a neural network model on images containing cats and dogs because we want the model to classify them. According to our definition of generalisation, given a held-out dataset, the model generalises well if it classifies most of these images correctly. Let us assume that against this held-out dataset the model achieves a good classification score. At test time then the model is presented with a new dataset of unseen images and this time it does not return a good classification score. *Is the model generalising well?* Probably not, although there is no straightforward answer to this question. In our generalisation assessment, in this simple example, we are making the assumption that training and test data are sampled from the same probability distribution. In reality, this assumption is often violated for many reasons, including the fact that training and test datasets might be too small and therefore introduce a significant selection bias due to the sample size.

Generalisation is a broad concept that should be regarded within the limitations of the task at hand, which are determined by factors including

model assumptions, datasets and evaluation measures, as we elaborate more in this chapter.

Generalisation Ability of a Generative Model

Assessing unsupervised learning is in general a challenging task and there is no perfect evaluation measure as the absence of labels makes this task prone to subjectivity. In our work we had to assess the quality of the samples of the compared models while at the same time evaluate their ability to move away from the training data to yield novel but plausible silhouettes. The *generalisation* ability that we are interested in, for a generative model, is the ability to generalise from the instances given in the training dataset. In other words, the model should be able to represent valid examples that are not seen in the training set while not straying away from the original domain of the object class. Thus, the generalisation ability is in direct contrast to *specificity*; if a model is so flexible to represent any shape, this comes at the cost of losing specificity (*e.g.*, we would not expect to sample chairs silhouettes from a model that is trained on a dataset of horses). Intuitively, a good compromise between generalisation and specificity is achieved when the probability density of the distribution learned by the model is spread out between and around the datapoints but still remaining mostly concentrated nearby the training instances.

In general, generative models offer more flexibility in assessing generalisation compared to discriminative models. We have mentioned that generalisation for a discriminative model can be assessed by means of classifying some unseen instances of a held-out dataset. In a generative model, we can use a similar approach by trying to "reconstruct" (generate) the unseen data and then use a similarity measure to compare the reconstructions and the unseen data, to quantify in this way how well the unseen data is reconstructed by the model. The additional flexibility of the generative model consists, for example, in the fact that if we train the model on images of cats and dogs and at test time we use instead a held-out dataset containing giraffes, a good generative model will return a low reconstruction score, indicating that it does not generalise to giraffes. In contrast, a two-class discriminative model trained on cats and dogs, when presented with images of giraffes will still classify them as either cats or dogs, making generalisation assessment harder.

A related desirable property that facilitates generalisation is *interpretability* (this is another broad concept with no single widely accepted definition, Lipton

(2018)). More specifically, in this work, we are interested in interpretability of the learned latent space. In simple terms, an interpretable latent space gives us an indication of where to sample from to generate plausible samples (as well as helping us to discriminate between unseen examples as to whether they belong to the manifold or not). In contrast, "black-box" methods, such as standard neural networks, can only be assessed by means of testing examples (which implies, in practice, that a large dataset is required at test time).

Some Challenges for Generalisation

Many factors can contribute to make good generalisation difficult to achieve. Insufficient training data can be a simple limiting factor. A training dataset consisting of very few datapoints might not adequately represent the true probability distribution of the data and as a consequence this might lead to poor learning for the model and poor generalisation.

Model's assumptions can also be limiting to generalisation. For instance, a model that is prone to memorise the training data and does not adequately capture and represent the intrinsic structure of the data is likely to achieve poor generalisation performance. In the case of neural networks, for example, Zhang et al. (2016) empirically demonstrated that these models are able to fit random noise. They tested various popular deep network models on a dataset in which all the labels were replaced with random ones. Most models achieved near to zero training error at classification. In order to achieve this, a neural network must use memorisation capabilities to some high degree. This result suggests that a neural network (with enough capacity, that is, with more parameters than there are training examples) can perfectly fit the data without much need of learning any patterns or features shared between the training datapoints, thus, making the content of what is memorised irrelevant. In contrast to these findings, subsequent studies suggest that although neural networks have the capacity to fit noise, they tend to prioritise simpler hypotheses such as shared patterns in the data. These works show qualitative differences in the behaviour of the network when fitting noise versus real data, for example, in terms of time to convergence in training, suggesting that exploiting patterns in the data is easier for a neural network than brute-force memorisation (Krueger et al., 2017; Arpit et al., 2017).

To what extent neural networks memorise training data is not well-understood, however, in general, these findings suggest that the prior assumptions of a model (either implicit model assumptions, *e.g.*, the structure of a neural network, or

explicitly enforced prior assumptions) play a key role for the generalisation ability.

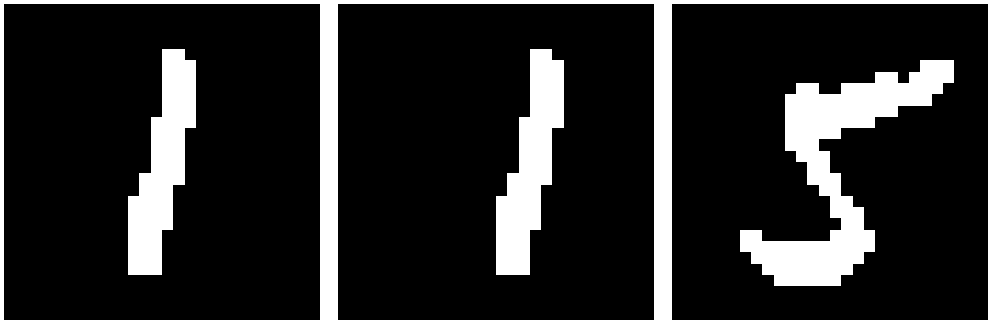
Another example where model's assumptions can have direct impact on generalisation is illustrated by the well-known GAN's *mode collapse problem* (Metz et al., 2017). This problem is mainly due to the way the GAN is optimised by means of playing a minimax game between the generator and discriminator networks. The problem occurs when the generator maps too many different input noise values to a very limited set of good high-dimensional samples (these are samples that are difficult for the discriminator to correctly identify as not coming from the training dataset), the generator in this way is able to deceive the discriminator very easily minimising the objective, and as a result, the generator's ability to generalise to novel samples is drastically reduced, because the generated data lacks of enough diversity. Some works in the literature try to reduce this issue by using a stabler objective (Metz et al., 2017) or by adding encoder-based regularisers (Che et al., 2017), and other empirical tricks (Salimans et al., 2016).

Our hypothesis is that the use of top-down priors can improve generalisation. Ideally, a model should be able to exploit prior knowledge about the data (for example, shape knowledge), however the more knowledge the model encapsulates about some specific type of data the less adaptable it becomes to other datasets. We believe that the smoothness assumption (a key feature of the GPDBN) represents a good compromise between enforcing a prior assumption about the data and maintaining good flexibility across different datasets. Exploring ways to include more prior knowledge seems to us a good line for future investigation.

Assessing Generalisation

Although the reconstruction error is normally used as a quantitative measure to assess generalisation, this approach does not simultaneously take into account specificity of the generated data (*i.e.*, the samples should be novel and at the same time belong in the same domain of the training data). To evaluate generalisation of a shape model, the fundamental questions that need to be answered are the following: *Can the model generalise to unseen shapes? Can the model generate quality novel data (in terms of affinity to the original data domain)?*

In order to answer these questions, the model's generated samples should be compared with ground-truth data from the real data distribution, unfor-



(a) (b) (c)

Figure 7.1: (a) A digit from the MNIST. (b) Same digit from the MNIST 3 pixels shifted to the right. (c) A different digit from the MNIST.

tunately though, this is impractical because this data is not available to us. Generalisation is thus assessed by means of measuring some reconstruction error against a small set of held-out data (*i.e.*, the model should try to generate the unseen data as closely as possible). However, this approach suffers from the above-mentioned problem of ignoring specificity (to exemplify this, we can imagine a model that is able to perfectly reconstruct the test input; however, when presented with a silhouette of a table the model would still achieve a perfect reconstruction score, independently of the fact that it was trained on horse silhouettes).

To take into account the model's specificity ability (in other words, the plausibility of the generated samples), in Chapter 5 we argued that a better way to assess generalisation is to reconstruct unseen data that is corrupted with salt-and-pepper noise, so that in order to achieve good results the model must also reject the implausibility introduced by the noise.

A related issue is the choice of a good similarity measure to compare the generated data with the test data. The SSIM (Wang et al., 2004) is a good measure in terms of structural information (compared to MSE or Cross-entropy, which estimate absolute errors) and it is also a popular choice in the literature (for these reasons we have used the SSIM in our experiments in Ch. 5). We believe, however, that current similarity measures (including the SSIM), in general, have many limitations. In the rest of this chapter we discuss this in more detail and outline some desirable features for a better similarity measure.

Measure	(a) vs. (b)	(a) vs. (c)
Mean Squared Error	9786.93	9786.93
Cross-Entropy	1891.57	1886.82
SSIM	0.69	0.63

Table 7.1: Scores for image (a) vs. (b) and (a) vs. (c) (Fig 7.1) calculated using common measures: MSE, CE and SSIM (the latter measures a similarity: the inverse of a distance, so the higher the score the higher the similarity).

Desirable Properties of a Generalisation Measure

Two widely used error measures in the literature are the mean squared error (MSE) and the cross-entropy (CE):

$$\text{MSE} = \frac{1}{M}\sum_{i=1}^{M}(s_i - t_i)^2 \ , \tag{7.1.1}$$

$$\text{CE} = -\sum_{i=1}^{M}[t_i \log(s_i) + (1 - t_i)\log(1 - s_i)] \ . \tag{7.1.2}$$

In Equations 7.1.1 and 7.1.2, t_i is the i-th pixel value of the test input and s_i is the i-th pixel of the model's sample (with M the number of pixels of the inputs). These measures fail to capture the structural diversity between images. This is because both the MSE and CE simply measure an absolute error between images that are merely regarded as a collection of pixel values, so that given a reference image it is easy to find permutations of the pixel values in the input sample while maintaining the same numerical score against the reference image. This is easy to see in Eqs. 7.1.1 and 7.1.2 as they consist of simple sums of uncorrelated pixel-wise errors between test and sample input.

Differently, the SSIM (Wang et al., 2004) is defined as follows:

$$\text{SSIM}(\boldsymbol{t}, \boldsymbol{s}) = \frac{(2\mu_t\mu_s + C_1)(2\sigma_{ts} + C_2)}{(\mu_t^2 + \mu_s^2 + C_1)(\sigma_t^2 + \sigma_s^2 + C_2)} \ , \tag{7.1.3}$$

which takes into account the pixel correlations; with $\boldsymbol{t}$ and $\boldsymbol{s}$ being the test and the sample image respectively, and C_1 and C_2 two small arbitrary positive

constants to avoid instability.

$$\mu_t = \frac{1}{M}\sum_{i=1}^{M} t_i \ , \tag{7.1.4}$$

$$\sigma_t^2 = \frac{1}{M}\sum_{i=1}^{M} (t_i - \mu_t)^2 \ , \tag{7.1.5}$$

$$\sigma_{ts} = \frac{1}{M}\sum_{i=1}^{M} (t_i - \mu_t)(s_i - \mu_s) \ . \tag{7.1.6}$$

Let us consider for example the three images in Fig. 7.1. The MSE and CE return the same score when the reference image (a) is measured against image (b) and against image (c). The SSIM is more robust against this problem, and as one would expect from a better measure, a lower score (indicating a higher distance between the images) is returned for reference image (a) against (c) compared to image (a) against (b). Quantitative details for this example can be found in Table 7.1.

We believe that a superior similarity measure would not only improve generalisation assessment but it could potentially inform the model's learning as well. In other words, if we know what makes a good similarity measure in the image space we can also exploit this knowledge to train the model to generalise better. In the simplest instance, an improved measure could be used to derive a better objective function for model optimisation (although, in order to use gradient descent based optimisation methods this would also require the measure to be differentiable).

The SSIM is a better measure compared to MSE and Cross-entropy as it takes into account the structural information in the input, however, it is not exempt from the problem of being sensitive to transformations of the input such as translation, scaling and rotation (there exists a lesser known extension of the SSIM to the complex wavelet domain, the *CW-SSIM* developed by Sampat et al. (2009), which should be more robust to small geometric distortion of the input, however it is not differentiable and does not solve the invariance problem to the extent that we would hope).

There exist plenty of similarity measures in the literature, reviewing them or investigating further on how to improve similarity measures is beyond the purpose of this work, but we believe that developing a better measure would be a good direction to follow for future work in order to improve a model's generalisation ability. We believe that some of the ideal characteristics that a similarity measure should have are: *differentiability*, so that the measure

could be incorporated in an objective function (which would ideally lead to closed-form analytical solutions for the model's objective); *robustness* in terms of invariance to translation, rotation and scaling of the input; finally, another desirable characteristic is a *low computational cost*. We are not aware of the existence of any such a measure in the literature with these features. Conceptually, at a very abstract level this could possibly be achieved through the inclusion of more specific prior knowledge about the data or other structural priors (such as convolutions). We acknowledge that having all these desirable characteristics in a single measure is an ambitious goal and future investigation and work is needed.

Chapter 8

Discussion and Conclusion

In this work we have presented the GPDBN, a model that combines the properties of a smooth, interpretable low-dimensional latent representation with a data specific non-Gaussian likelihood function (for silhouette images). The model fully propagates and captures uncertainty in its estimates allowing it to make efficient use of the data and provides an interpretable latent space facilitating good sampling. Moreover, the model is trained end-to-end with the same complexity as a standard feed-forward neural network by minimising a single objective function, and can learn from very little data as well as scaling to larger datasets linearly by using mini-batching. We have shown both quantitatively and qualitatively that the proposed model performs better or on par with the best shape models while at the same time introducing a smooth and low-dimensional latent representation with associated uncertainty that facilitates easy synthesis of data.

In Chapter 6 we have presented some alternative dropout architectures for uncertainty propagation throughout the model. Our results suggest the use of Concrete units is the best approach to introduce stochasticity in the network and propagate uncertainty through the GPDBN.

In Chapter 7 we have argued that generalisation is a subtle concept that needs to be regarded in the context of the task at hand and within the limits of assumptions that are made. The common way of assessing the generalisation performance of a model by merely reconstructing some test data can lead to misleading results, we have proposed instead the use of test data corrupted with noise to avoid the problem of the model simply replicating the test input to achieve a high generalisation score.

Model's Limitations and Future Work

An issue that is common to deep learning models is the fact that in general it is not clear how to choose the right number of layers and units in the architecture. This limitation also affects the GPDBN model. Currently, model's specifications such as the number of layers and units must be specified manually based on empirical considerations, and as we have shown in Section 5.9, where we have experimented with different architectures, it is possible to fine-tune the model to achieve slight performance gains (though this is a time-consuming process). We think that this issue could be addressed using a Bayesian approach with sparsity inducing priors to prune the network and to find the optimal architecture automatically. This needs further investigation and we leave it as open opportunity for future work.

We have shown in Section 4.8 that while adding convolutional layers allows the GPDBN model to scale to larger input images it also increases the complexity of the model and introduces some artefacts in the output. Mini-batching (Sec. 4.6), on the other hand, allows the model to scale to larger datasets but uses a biased objective estimator. Finding better ways to allow the GPDBN to scale to larger images and datasets is still an open problem.

Besides the desirable properties such as smooth and interpretable manifold, uncertainty propagation and easy training with a single objective function, which, as we have shown in our work, can be combined into a single model, it is very important for a generative model to generalise well to novel examples. We have argued that good generalisation strongly depends on the similarity measure, especially because the characteristics of a good measure could be exploited to inform the model itself. For this reason, we believe that working on improving similarity measures is another good direction for future work.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Available from: https://www.tensorflow.org/.

Andre, A. and Saito, S., 2011. Single-view Sketch Based Modeling. *ACM Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*.

Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M.S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y. and Lacoste-Julien, S., 2017. A Closer Look at Memorization in Deep Networks. *International Conference on Machine Learning*.

Bengio, Y., Courville, A. and Vincent, P., 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), pp.1798–1828.

Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H., 2007. Greedy Layer-Wise Training of Deep Networks. *Advances in Neural Information Processing Systems*.

Bengio, Y. and LeCun, Y., 2007. *Scaling Learning Algorithms toward AI*.

Borenstein, E., Sharon, E. and Ullman, S., 2004. Combining Top-Down and Bottom-Up Segmentation. *Conference on Computer Vision and Pattern Recognition Workshop*.

Campbell, N.D.F. and Kautz, J., 2014. Learning a Manifold of Fonts. *ACM Transactions on Graphics*, 33(4), pp.91:1–91:11.

Carreira-Perpiñán, M.A. and Hinton, G.E., 2005. On Contrastive Divergence Learning. *International Conference on Artificial Intelligence and Statistics Workshop*.

Che, T., Li, Y., Jacob, A.P., Bengio, Y. and Li, W., 2017. Mode Regularized Generative Adversarial Networks. *International Conference on Learning Representations*.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I. and Abbeel, P., 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in Neural Information Processing Systems*.

Cremers, D., 2006. Dynamical Statistical Shape Priors for Level Set-based Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8), pp.1262–1273.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4), pp.303–314.

Damianou, A. and Lawrence, N., 2013. Deep Gaussian Processes. *International Conference on Artificial Intelligence and Statistics Workshop*.

Di Martino, A., Bodin, E., Ek, C.H. and Campbell, N., 2018. Gaussian Process Deep Belief Networks: A Smooth Generative Model of Shape with Uncertainty Propagation. *Asian Conference on Computer Vision*.

Di Martino, A., Bodin, E., Ek, C.H. and Campbell, N., 2019. GPDBN Code. Available from: https://github.com/zeis/deepbelief.

Elgammal, A. and Lee, C.S., 2004. Inferring 3D Body Pose from Silhouettes Using Activity Manifold Learning. *Conference on Computer Vision and Pattern Recognition*.

Elhabian, S.Y. and Whitaker, R.T., 2017. ShapeOdds: Variational Bayesian Learning of Generative Shape Models. *Conference on Computer Vision and Pattern Recognition*.

Eslami, S.M.A., Heess, N., Williams, C.K.I. and Winn, J., 2014. The Shape Boltzmann Machine: A Strong Model of Object Shape. *International Journal of Computer Vision*, 107(2), pp.155–176.

Eslami, S.M.A. and Williams, C.K.I., 2012. A Generative Model for Parts-Based Object Segmentation. *Advances in Neural Information Processing Systems*.

Fei-Fei, L., Fergus, R. and Perona, P., 2004. Learning Generative Visual Models from few Training Examples: an Incremental Bayesian Approach Tested on 101 Object Categories. *Conference on Computer Vision and Pattern Recognition Workshop*.

Ferrari, V., Jurie, F. and Schmid, C., 2010. From Images to Shape Models for Object Detection. *"International Journal of Computer Vision*, 87(3), pp.284–303.

Gal, Y., Hron, J. and Kendall, A., 2017. Concrete Dropout. *Advances in Neural Information Processing Systems*.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative Adversarial Nets. *Advances in Neural Information Processing Systems*.

Grosgeorge, D., Petitjean, C., Dacher, J.N. and Ruan, S., 2013. Graph Cut Segmentation with a Statistical Shape Model in Cardiac MRI. *Computer Vision and Image Understanding*, 117(9), pp.1027–1035.

Gu, L. and Kanade, T., 2008. A Generative Shape Regularization Model for Robust Face Alignment. *European Conference on Computer Vision*.

Hinton, G.E., 2012. *A Practical Guide to Training Restricted Boltzmann Machines*.

Hinton, G.E. and Salakhutdinov, R.R., 2006. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786), pp.504–507.

Hornik, K., Stinchcombe, M. and White, H., 1989. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5), pp.359–366.

Kalogerakis, E., Chaudhuri, S., Koller, D. and Koltun, V., 2012. A Probabilistic Model for Component-based Shape Synthesis. *ACM Transactions on Graphics*, 31(4), pp.55:1–55:11.

Kingma, D.P. and Ba, J., 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*.

Kingma, D.P. and Welling, M., 2013. Auto-Encoding Variational Bayes. *International Conference on Learning Representations*.

Kirillov, A., Gavrikov, M., Lobacheva, E., Osokin, A. and Vetrov, D., 2016. Deep Part-Based Generative Shape Model with Latent Variables. *British Machine Vision Conference*.

Krueger, D., Ballas, N., Jastrzebski, S., Arpit, D., Kanwal, M.S., Maharaj, T., Bengio, E., Fischer, A. and Courville, A., 2017. Deep Nets Don't Learn via Memorization. *International Conference on Learning Representations*.

Lawrence, N. and Quiñonero-Candela, J., 2006. Local Distance Preservation in the GP-LVM Through Back Constraints. *International Conference on Machine Learning*.

Lawrence, N.D., 2005. Probabilistic Non-linear Principal Component Analysis with Gaussian Process Latent Variable Models. *The Journal of Machine Learning Research*, 6, pp.1783–1816.

LeCun, Y., Cortes, C. and Burges, C.J.C., 1998. The MNIST Database of Handwritten Digits. Available from: http://yann.lecun.com/exdb/mnist/.

Li, W., Viola, F., Starck, J., Brostow, G.J. and Campbell, N.D.F., 2016. Roto++: Accelerating Professional Rotoscoping using Shape Manifolds. *ACM Transactions on Graphics*, 35(4), pp.62:1–62:15.

Li, Y., Gu, L. and Kanade, T., 2009. A Robust Shape Model for Multi-view Car Alignment. *Conference on Computer Vision and Pattern Recognition*.

Lipton, Z.C., 2018. The Mythos of Model Interpretability. *Queue*, 16(3), pp.30:31–30:57.

M. K. Titsias and N. D. Lawrence, 2010. Bayesian Gaussian Process Latent Variable Model. *International Conference on Artificial Intelligence and Statistics*.

Maddison, C.J., Mnih, A. and Teh, Y.W., 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *International Conference on Learning Representations*.

Metz, L., Poole, B., Pfau, D. and Sohl-Dickstein, J., 2017. Unrolled Generative Adversarial Networks. *International Conference on Learning Representations*.

Patenaude, B., Smith, S.M., Kennedy, D.N. and Jenkinson, M., 2011. A Bayesian Model of Shape and Appearance for Subcortical Brain Segmentation. *Neuroimage*, 56(3), pp.907–922.

Prisacariu, V. and Reid, I., 2011. Nonlinear Shape Manifolds as Shape Priors in Level Set Segmentation and Tracking. *Conference on Computer Vision and Pattern Recognition*.

Prisacariu, V.A. and Reid, I.D., 2012. PWP3D: Real-Time Segmentation and Tracking of 3D Objects. *International Journal of Computer Vision*, 98(3), pp.335–354.

Rasmussen, Carl Edward and Williams, Christopher K.I., 2006. *Gaussian Processes for Machine Learning*. The MIT Press.

Reinbacher, C., Ruther, M. and Bischof, H., 2010. Pose Estimation of Known Objects by Efficient Silhouette Matching. *International Conference on Pattern Recognition*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X., 2016. Improved Techniques for Training GANs. *Advances in Neural Information Processing Systems*.

Sampat, M.P., Wang, Z., Gupta, S., Bovik, A.C. and Markey, M.K., 2009. Complex Wavelet Structural Similarity: A New Image Similarity Index. *IEEE Transactions on Image Processing*, 18(11), pp.2385–2401.

Scikit-image, 2019. SSIM Implementation. https://scikit-image.org/ [Accessed: 2019-05-21].

Smolensky, P., 1986. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*.

Snoek, J., Adams, R.P. and Larochelle, H., 2012. Nonparametric Guidance of Autoencoder Representations Using Label Information. *The Journal of Machine Learning Research*, 13(1), pp.2567–2588.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1), pp.1929–1958.

Taylor, C., Twining, C. and Davies, R., 2008. *Statistical Models of Shape*. Springer.

Tieleman, T., 2008. Training Restricted Boltzmann Machines Using Approximations to the Likelihood Gradient. *International Conference on Machine Learning*.

Toshev, A., Makadia, A. and Daniilidis, K., 2009. Shape-based Object Recognition in Videos Using 3D Synthetic Object Models. *Conference on Computer Vision and Pattern Recognition*.

Toshev, A., Taskar, B. and Daniilidis, K., 2012. Shape-based Object Detection via Boundary Structure Segmentation. *International Journal of Computer Vision*, 99(2), pp.123–146.

Tosi, A., Hauberg, S., Vellido, A. and Lawrence, N.D., 2014. Metrics for Probabilistic Geometries. *Uncertainty in Artificial Intelligence*.

Tsogkas, S., Kokkinos, I., Papandreou, G. and Vedaldi, A., 2015. Deep Learning for Semantic Part Segmentation with High-Level Guidance. *arXiv preprint arXiv:1505.02438*.

Turmukhambetov, D., Campbell, N.D.F., Goldman, D.B. and Kautz, J., 2015. Interactive Sketch-Driven Image Synthesis. *Computer Graphics Forum*, 34(8), pp.130–142.

Wang, Z., Bovik, A.C., Sheikh, H.R. and Simoncelli, E.P., 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4), pp.600–612.

Wikipedia, 2019. Generalization. https://en.wikipedia.org/wiki/Generalization [Accessed: 2019-03-19].

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X. and Xiao, J., 2015. 3D ShapeNets: A Deep Representation for Volumetric Shapes. *Conference on Computer Vision and Pattern Recognition*.

Yang, Y. and Ramanan, D., 2011. Articulated Pose Estimation with Flexible Mixtures-of-Parts. *Conference on Computer Vision and Pattern Recognition.*

Yemez, Y. and Schmitt, F., 2004. 3D Reconstruction of Real Objects with High Resolution Shape and Texture. *Image and Vision computing*, 22(13), pp.1137–1153.

Zhang, C., Bengio, S., Hardt, M., Recht, B. and Vinyals, O., 2016. Understanding Deep Learning Requires Rethinking Generalization. *arXiv preprint arXiv:1611.03530.*